
Sharif University
Electrical Engineering School

---

Communication Systems Project

# IMPLEMENTING A DIGITAL COMMUNICATION SYSTEM

Armin Panjehpour
arminpp1379@gmail.com

Supervisor(s): Dr. Pakravan
Sharif University, Tehran, Iran

09/02/2022

# Part.1 - Introduction

In this project, we will implement a digital communication system with decoder and encoder as you can see below:

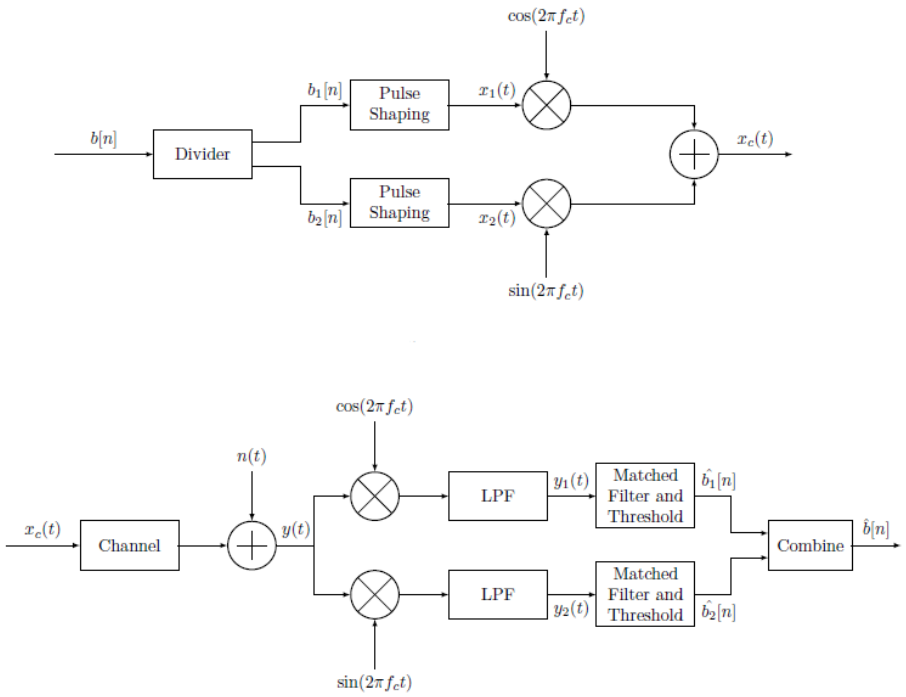


Figure 1: Structure of the system

To make the project more simple, we just consider binary sequences.

# Part.2 - Implementation of blocks

**Part.2.1 - Divide | Combine Blocks** b[n] is a sequence of n (n is even) bits which is the digital signal we want to transmit. Divider, as the first element of the system, will divide the sequence in two equal parts. It puts the even bits in $b_1[n]$ and the odd bits in $b_2[n]$. The inverse function of Divider is Combiner which gets 2 sequences with length of n/2 and gives a sequence with a length of n in the output ($\hat{b}_1[n]$ will be the even bits and $\hat{b}_2[n]$ will be the odd bits of the output sequence). You can see their implementation below:

```
// Divider
    function [b1, b2] = Devide(b)
        b1 = b(1:2:end);
        b2 = b(2:2:end);
    end
```

```
// Combiner
    function b = Combine(b1,b2)
        b = zeros(1,length(b1)+length(b2));
        b(1:2:end) = b1;
        b(2:2:end) = b2;
    end
```

you can see an example of these functions below:

```
inputSignal =
     0     0     1     0     0     0     1     1     1     1

b1 =
     0     1     0     1     1

b2 =
     0     0     0     1     1

bprime1 =
     1     1     1     0     1

bprime2 =
     1     0     1     0     1

bprime =
     1     1     1     0     1     1     0     0     1     1
```

Figure 2: Divider | Combiner Results

**Part.2.2 - Pulse Shaping Block** As we know, we have to do line coding to send digits as a waveform in our system. There are many different line codes like On-Off system, Polar system, Bipolar system, etc. We will discuss on the pulse shapes further. In PulseShaping function, we will produce a pulse with length of 10ms with a sampling rate of 1MHz specifically for digit 1 and another pulse with the same length specifically for digit 0. For example, for Polar system, we will send a pulse, p(t), for digit 1 and a pulse, -p(t), for digit 0:

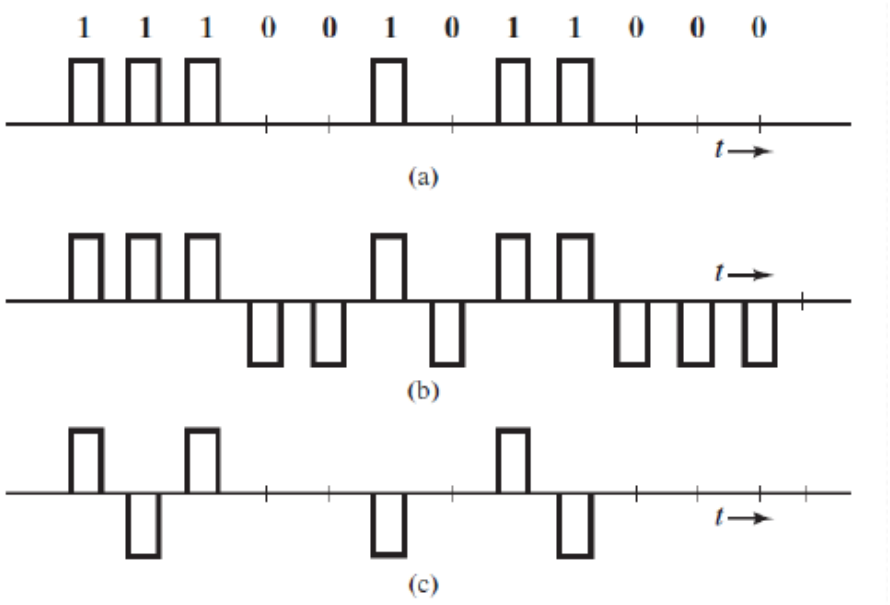


Figure 3: a) On-Off System || b) Polar System || c) Bipolar System

PulseShaping function implementation:

```
// Line Coding

    function pulse = PulseShaping(inputSequence, zeroPulse, onePulse)
        pulse = ((inputSequence.').*onePulse) + ...
            ((1-inputSequence.').*zeroPulse);
        pulseSize = size(pulse);
        pulse = reshape(pulse.',[1 pulseSize(1)*pulseSize(2)]);
    end
```

For example, if we give 1011000001 as a input to this function, with p(t) being an square pulse, the analog waveform with a polar system would be like this:

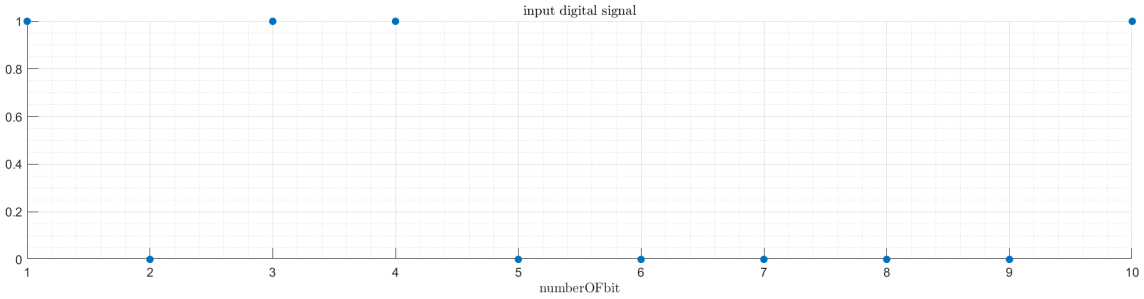


Figure 4: Digital Input Signal

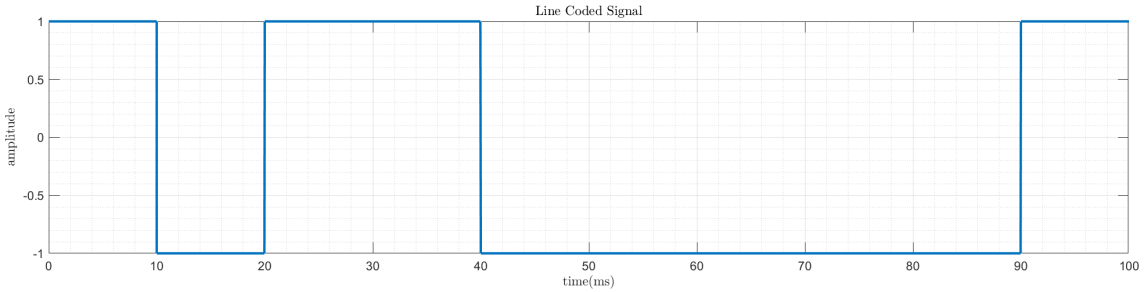


Figure 5: Analog Waveform

**Part.2.3 - Analog Modulation** AnalogMod box will modulate the input signals by multiplying them to cosine functions and give $x_c(t)$ as sum of these two modulated signals. You can see the implementation of this function below:

```
// Analog Modulation

    function xc = AnalogMod(pulse1,pulse2,fs,fc)
        t = 0:1/fs:(length(pulse1)/fs)-(1/fs);
        xc = pulse1.*cos(2*pi*fc.*t) + pulse2.*sin(2*pi*fc.*t);
    end
```

An example of analog modulation of a polar line coding system:

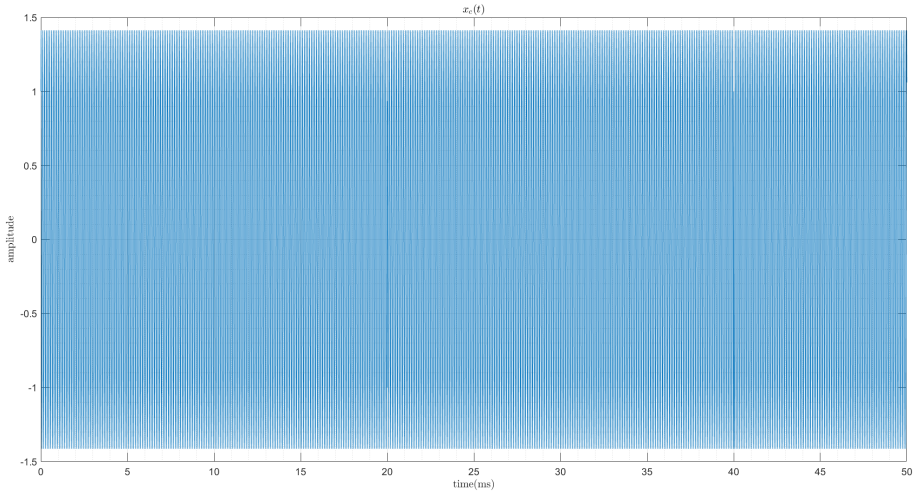


Figure 6: Analog Modulation

**Part.2.4 - Channel** The channel is just a simple Bandpass filter with center frequency of the channel ($f_c$) and bandwidth of the channel ($BW_c$):

```
// Channel

        function channelOutput = Channel(xc,fs,fcCh,BWch)
            channelOutput = bandpass(xc,[fcCh-BWch/2,fcCh+BWch/2],fs);
        end
```

Signal in Part 2.3 (Fig.6) transmitted from the channel:

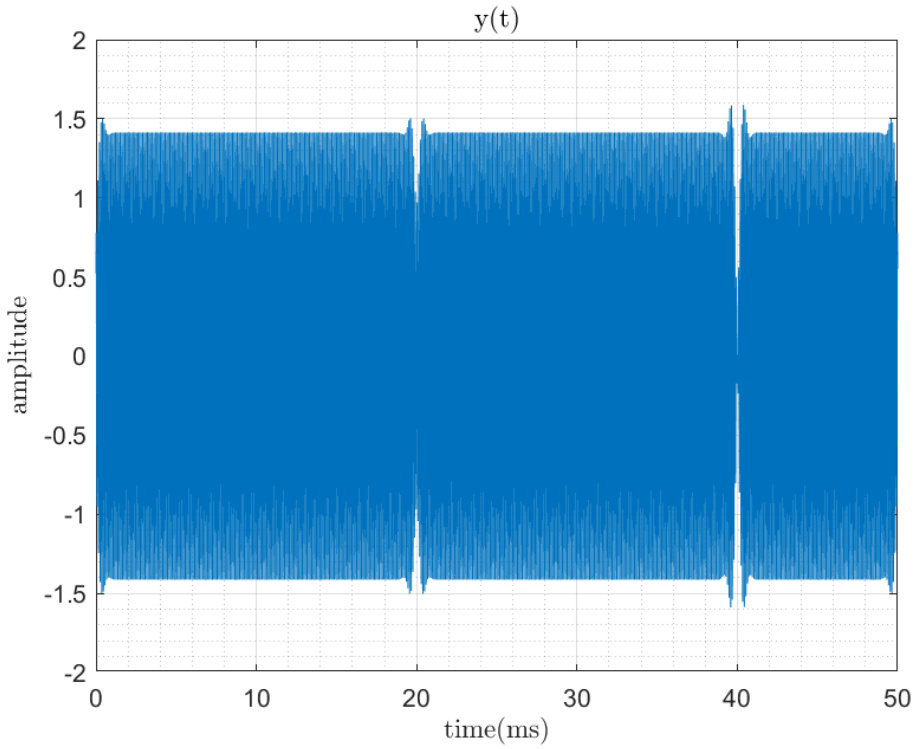


Figure 7: Channel Output

**Part.2.5 - Analog Demodulation** This block will demodulate the signal modulated before passing through the channel by multiplying to cosine functions and at the end, it will apply a low pass filter to get the frequency components around the baseband:

```
// Analog Demodulation

    function [y1, y2] = AnalogDemod(xc,fs,BWs,fc)
        t = 0:1/fs:(length(xc)/fs)-(1/fs);
        y1 = lowpass(xc.*cos(2*pi.*fc.*t),BWs,fs);
        y2 = lowpass(xc.*sin(2*pi.*fc.*t),BWs,fs);
    end
```

As you can see in (Fig.8), we can find out from the amplitude of demodulated signals that what was the transmitted digital bit:

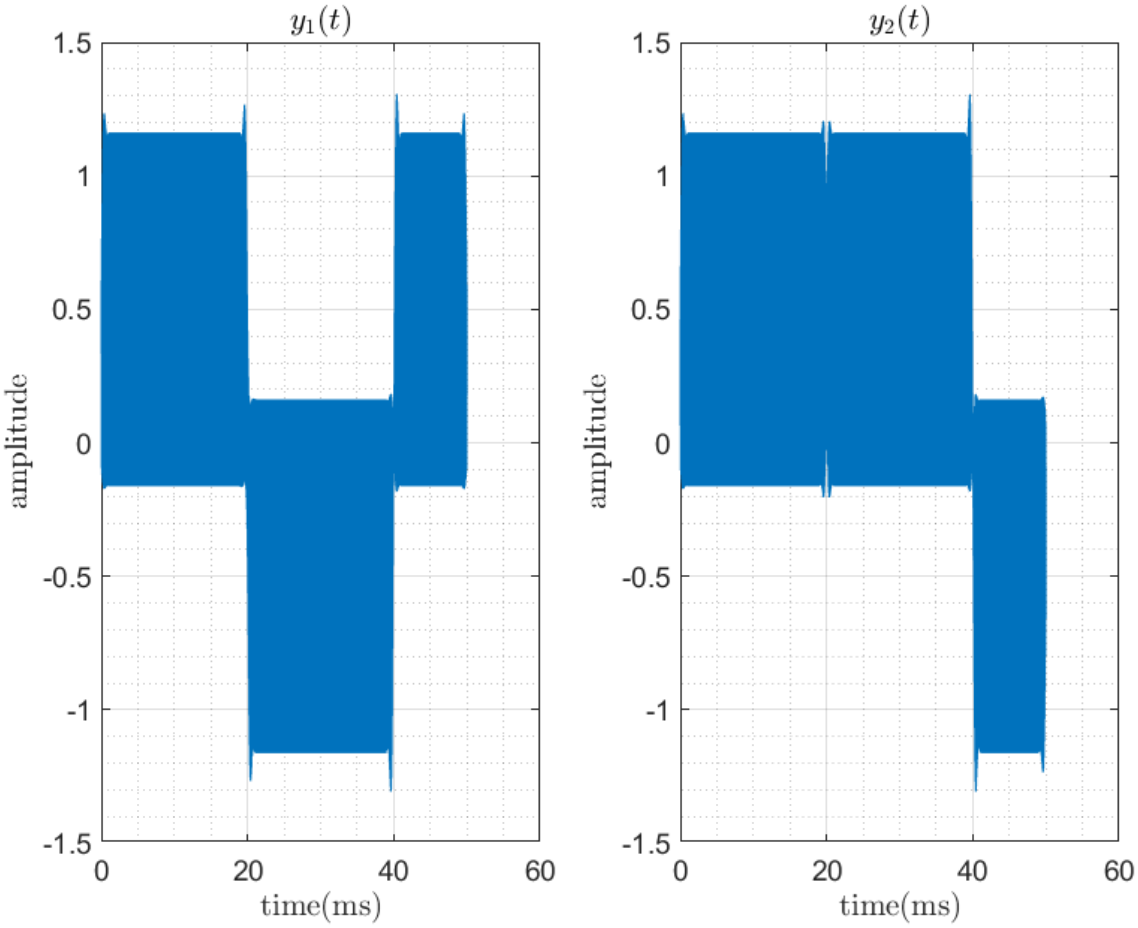


Figure 8: Demodulated Signals

**Part.2.6 - Matched Filter** As you saw in (Fig.8), a filter is needed to export the transmitted bits from the demodulated signals. Matched filters will do this for us. Match filter is a filter which Convolves the demodulated signal with pulses for bit 1 and bit 0 and compare them and then decides what was the transmitted bit:

```
// Matched Filter

function [raw0 raw1 Reconstructed] = ...
MatchedFilt(demodulatedSignal, zeroPulse, onePulse)
    Reconstructed = zeros(1,length(demodulatedSignal)/length(zeroPulse));
    raw0 = zeros(1,length(demodulatedSignal)/length(zeroPulse));
    raw1 = zeros(1,length(demodulatedSignal)/length(zeroPulse));
    r0 = conv(demodulatedSignal,zeroPulse);
    r1 = conv(demodulatedSignal,onePulse);

    for i=1:length(Reconstructed)
        raw0(i) = r0(i*length(zeroPulse));
        raw1(i) = r1(i*length(zeroPulse));
        if(r1(i*length(zeroPulse)) > r0(i*length(zeroPulse)))
            Reconstructed(i) = 1;
        else
            Reconstructed(i) = 0;
        end
    end
end
```

The matched filter result will be 2 sequences of bits which by giving them to Combine block, we can have the reconstructed digital signal.

# Part.3 - Transmitting a sequence of random binary numbers

Characteristics of the system:

| Variable | Value |
|---|---|
| Sampling Rate | 1MHz |
| Pulse Length | 10ms |
| Fc | 10KHz |
| Fcch | 10KHz |
| BWch | 1KHz |

**Part.3.1 - PAM Modulation** Here, p(t) is a simple square pulse with amplitude 1 for sending bit 1 and amplitude -1 for sending bit 0.

**Part.3.1.A - Simulation** Here we consider the input messages to be 10 bits digital signals for more clear results. At first, a 10 bits signal of 0,1 is created using the randi function in MATLAB:

```
lengthSignal = 10; % arbitrary | should be even
inputSignal = randi([0 1], 1, lengthSignal);
```

Pulses for sending 0,1 bits are implemented as you can see below:

```
// tPulse is 10ms and Fs is 1MHz
onePulse = ones(1,tPulse*Fs);
zeroPulse = -1*ones(1,tPulse*Fs);
```

The input signal which is created is: 0011011010 :

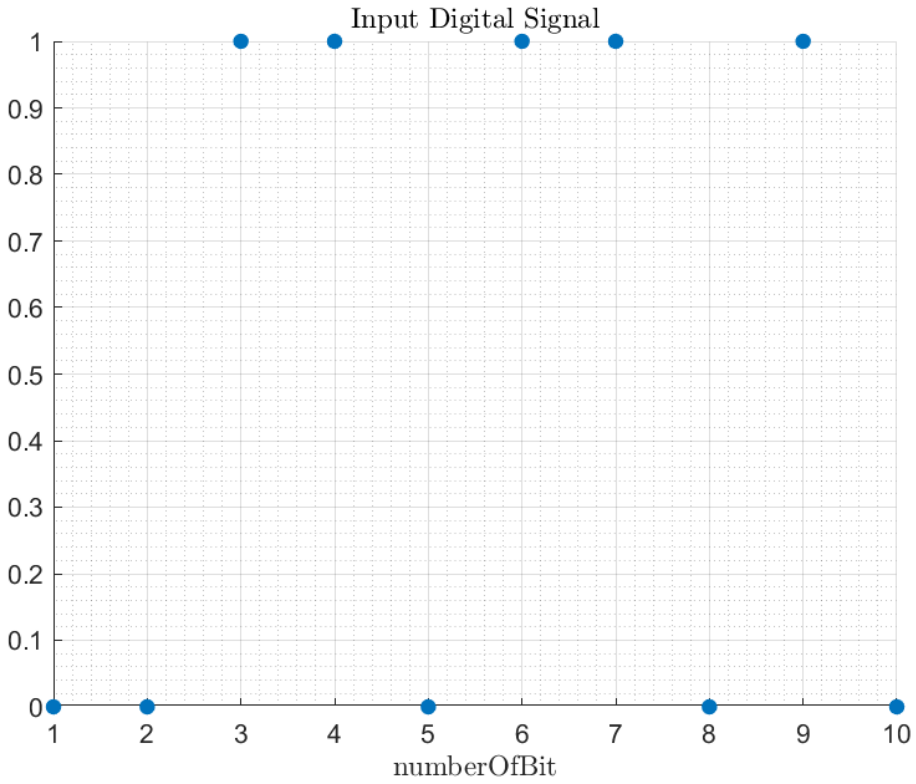


Figure 9: Input Digital Signal - PAM

Then, the input is passed through divider block as you saw the implementation in previous parts and here‘s the result:

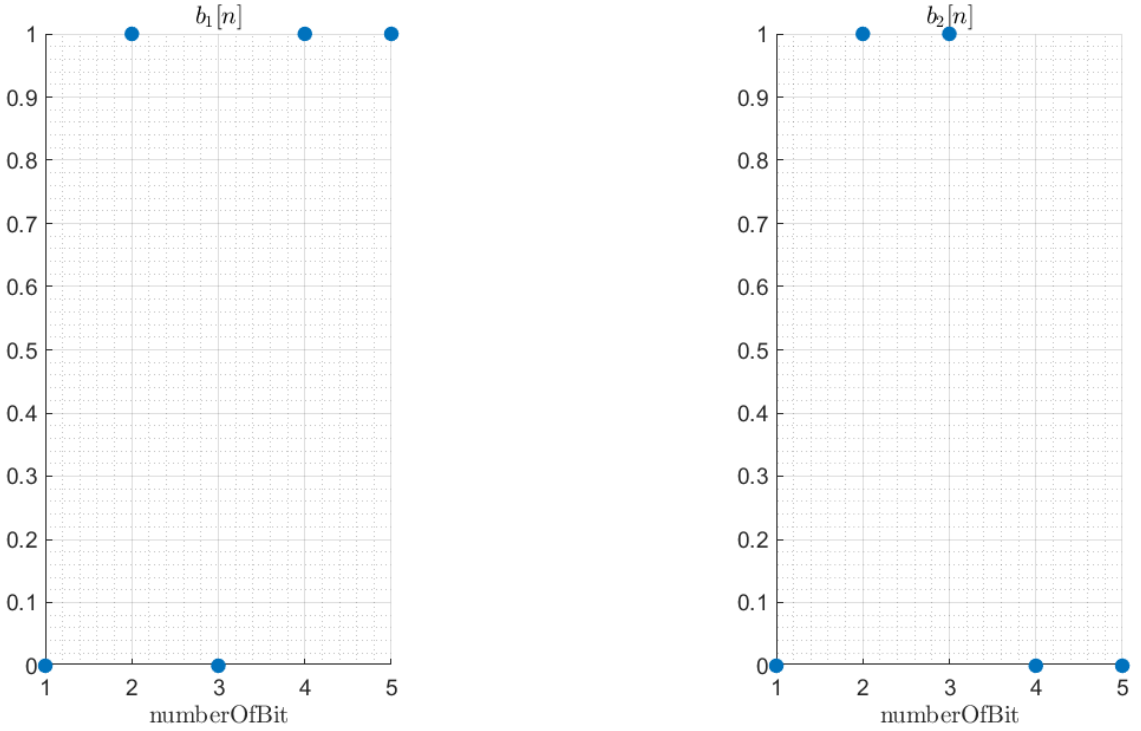


Figure 10: Divided Signals - PAM

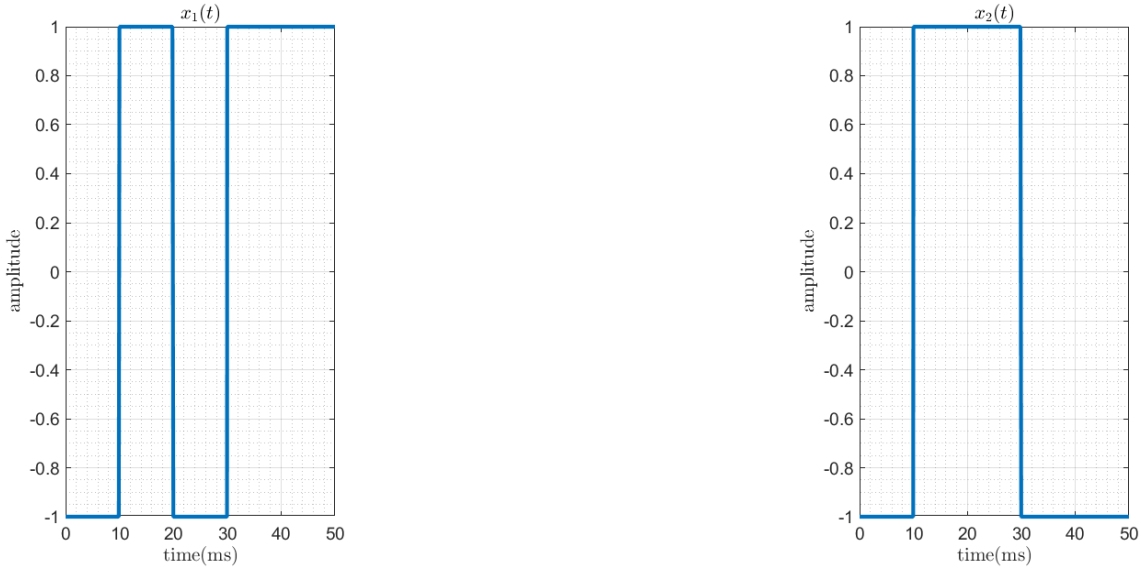


Figure 11: Analog Waveform - PAM

Then, each divided signal is passed through the PulseShaping block and an analog waveform corresponding to each of two sequences of bits is created as you can see in Fig.11.

After that, each analog waveform goes through the analog modulation. The modulated signal is:

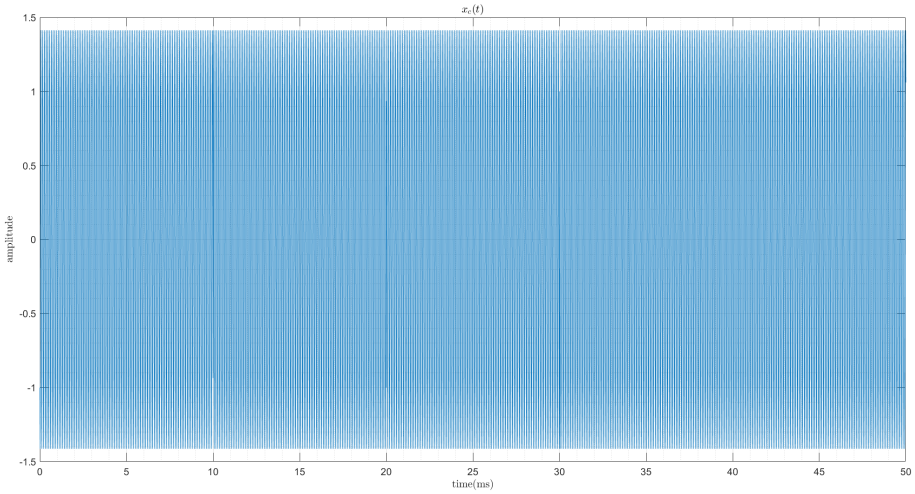


Figure 12: Modulated Signal - PAM

Then, the signal will pass through the channel and a noise will be added to it. In this part we don‘t have noise ant n(t) equals to zero. Here‘s the output of the channel:

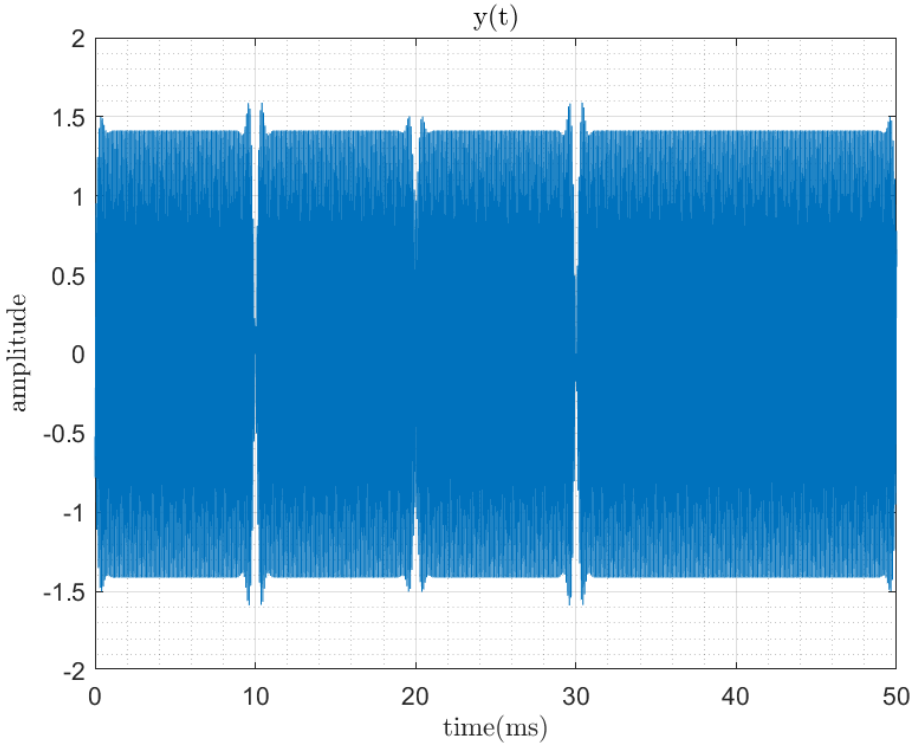


Figure 13: Channel‘s output - PAM

Next, the output of the channel will be demodulated using AnalogDemod function:

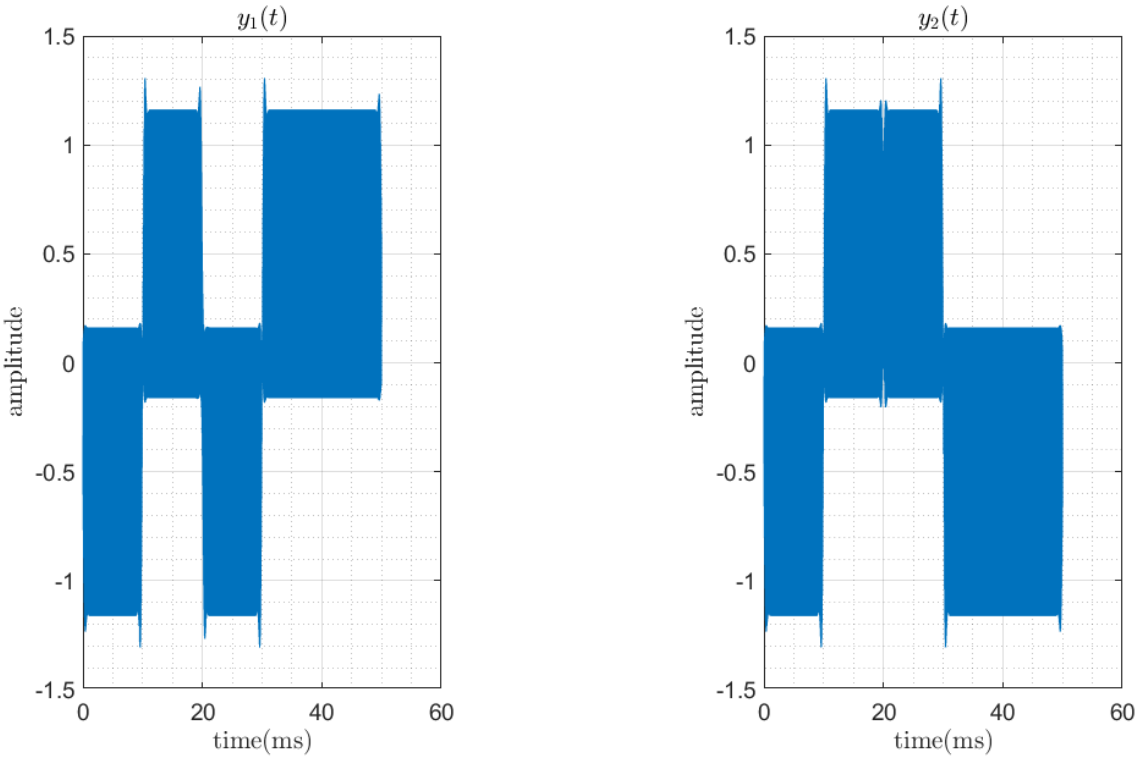


Figure 14: Demodulated Signals - PAM

As you can see, the demodulated signals just need a Matched filter to export the input sequence from them. So the Matched filters outputs are:

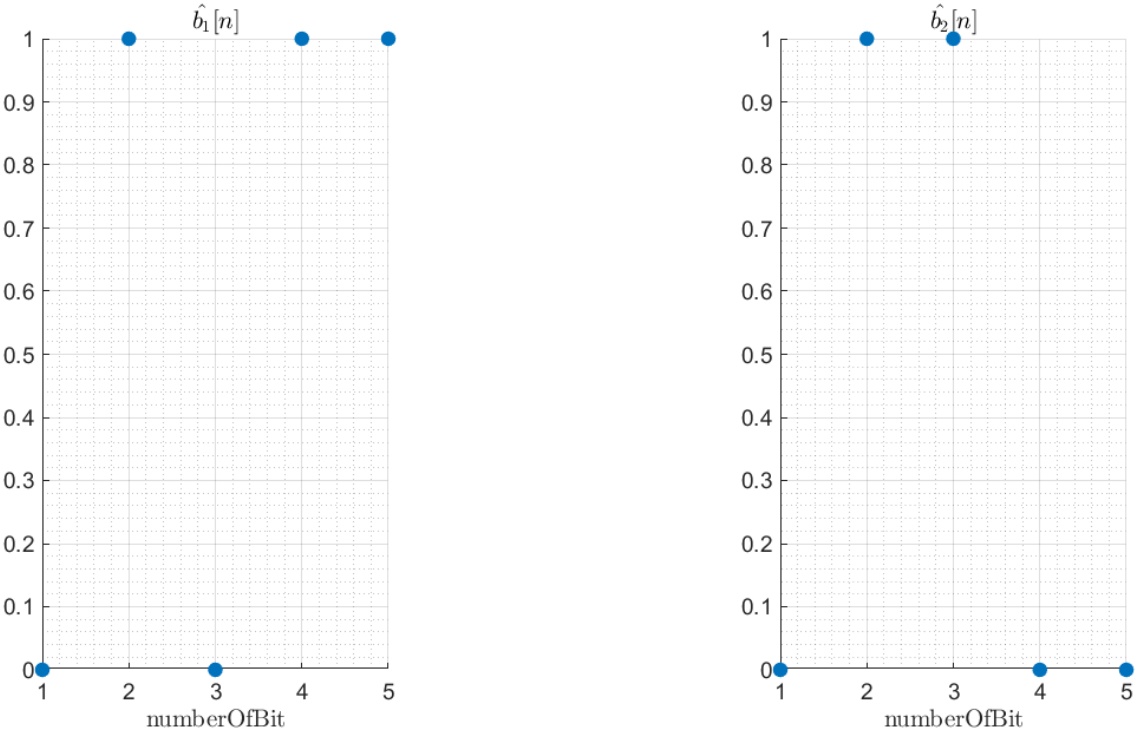


Figure 15: Outputs of Matched Filters - PAM

Then using Combine function, we will have the reconstructed sequence which equals to the input signal:

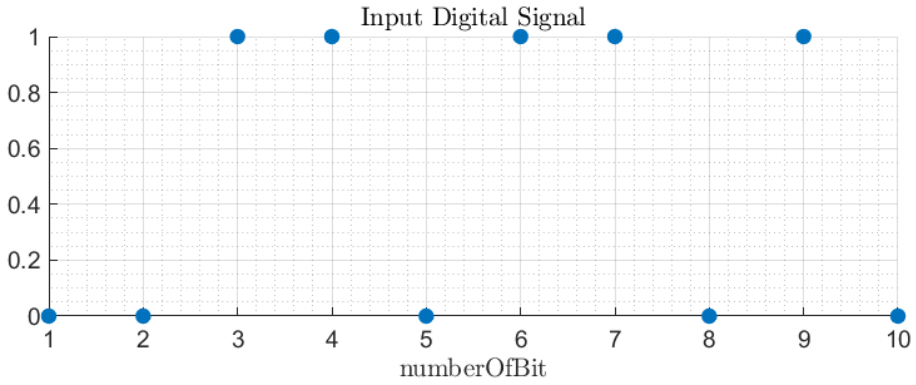


Figure 16: Input Signal - PAM

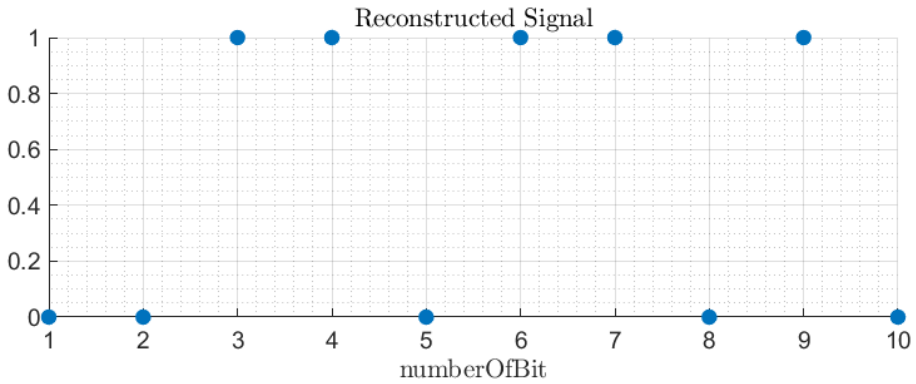


Figure 17: Outputs of Matched Filters - PAM

**Part.3.1.B - Adding Noise** As we saw, the n(t) which is the noise will be added to the output of the channel. In previous part, n(t) was zero and the system was ideal. That's why we get 100% accuracy in signal reconstruction. Now we assume n(t) is an white Gaussian noise which will be added to the output of the channel (AWGN) with a mean of 0. We create this noise using normrnd function in MATLAB:

```
//Noise
noise = normrnd(mu,sigma,1,length(y));
```

normrnd will create a vector of independent random numbers following the normal distribution with a specific mean and standard deviation (SD). Now we calculate the probability of having error in reconstruction for SD from 1 to 100. For each SD, we run the simulation 5 times and calculated the average error and then we did the averaging on all 10 bits which lead us to this result:

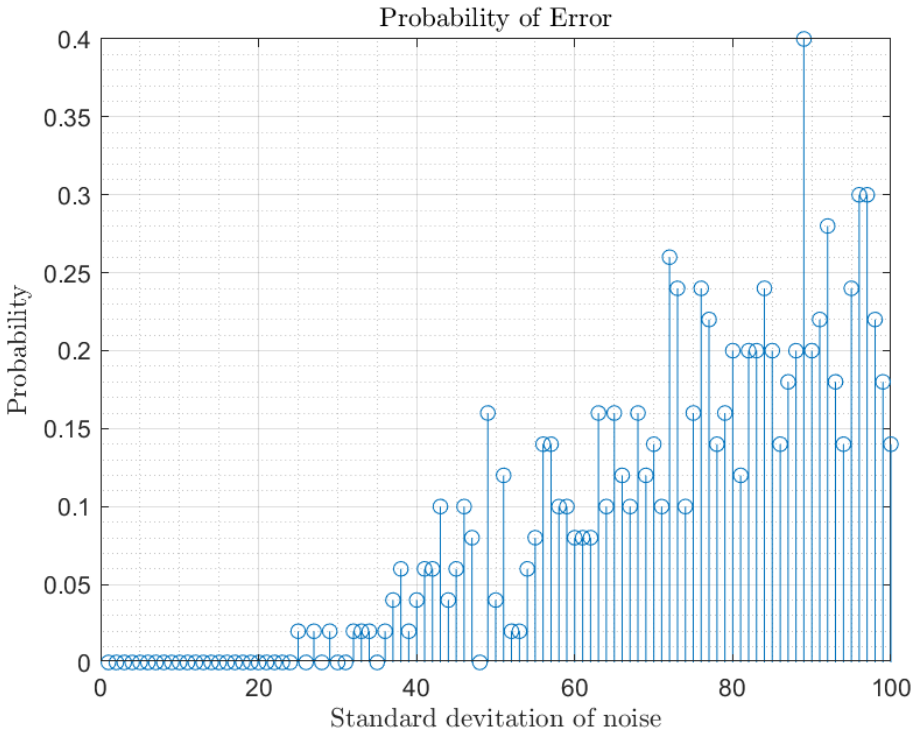


Figure 18: Error Probability - PAM

As we can see, by increasing the SD, the probability of error happening is increasing on average.

**Part.3.1.C - Signal Constellation** Using these 6 SDs, [10 28 32 40 73 99], we plot the 2D Matched filter of the outputs of two matched filters:

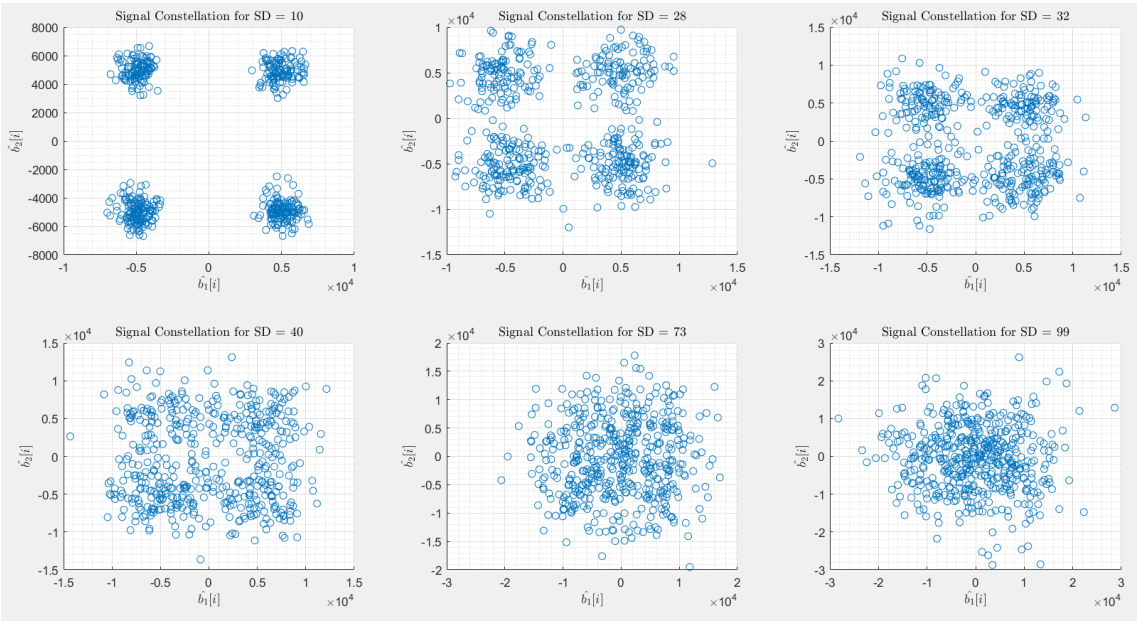


Figure 19: Signal Constellation of PAM

we can see, for less SDs, the points are all closer to (-+1,-1+) and this shows that we have less error for less SDs. As the SD increases, the more distributed the points will be.

**Part.3.2 - PSK Modulation** Now we repeat all of the analysis in part.3.1 for PSK modulation, the only difference is in the p(t) format. Here p(t) is a cos(t) function with a frequency of 500 Hz:

```
// pulse
f = 500; % in Hz
onePulse = cos(2*pi*f*(0:1/Fs:tPulse-1/Fs));
zeroPulse = -cos(2*pi*f*(0:1/Fs:tPulse-1/Fs));
```

**Part.3.2.A - Simulation** The input signal with a length of 10: 1101001101 :

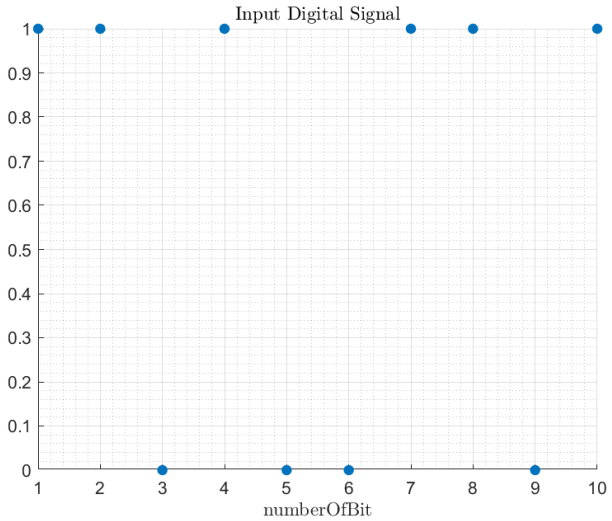


Figure 20: Input Digital Signal - PSK

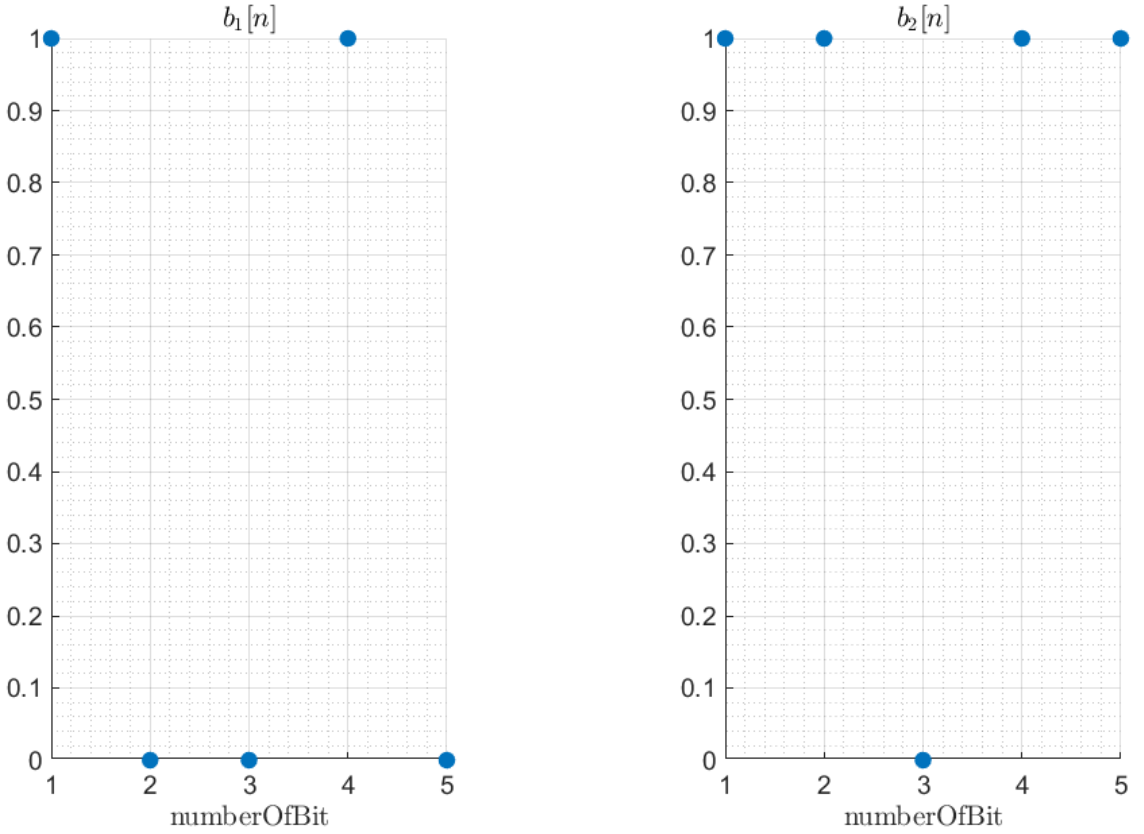


Figure 21: Divided Signals - PSK

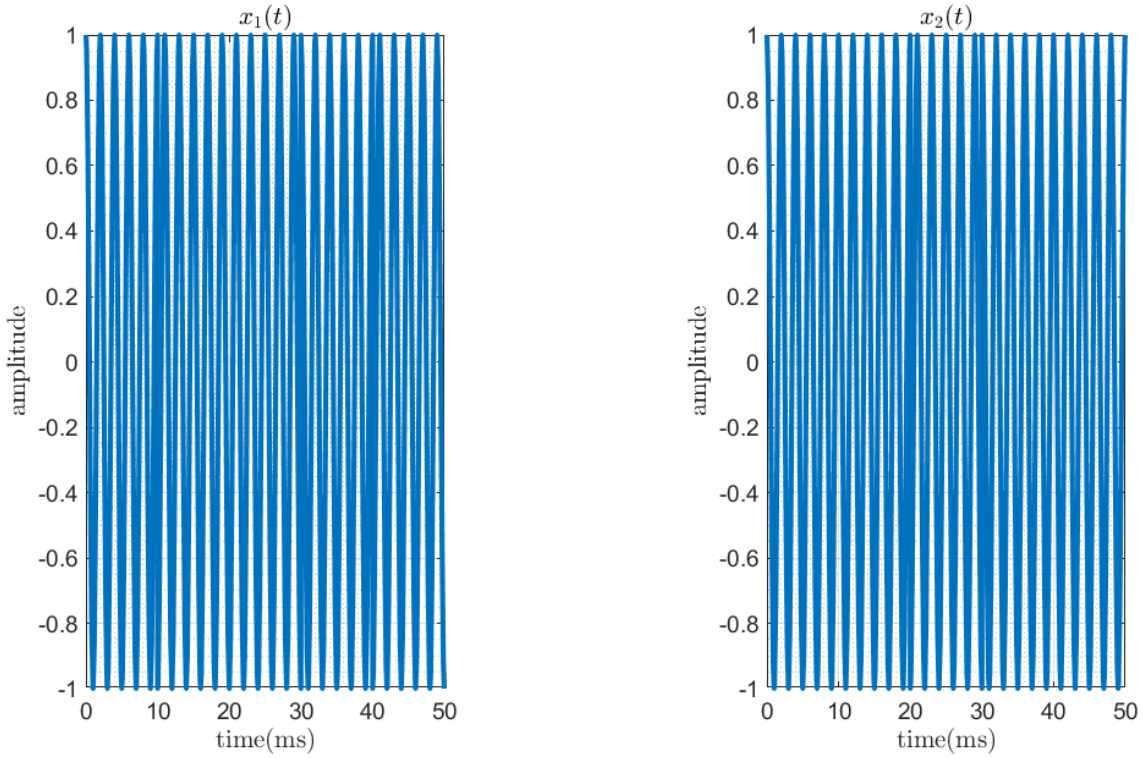


Figure 22: Analog Waveforms - PSK

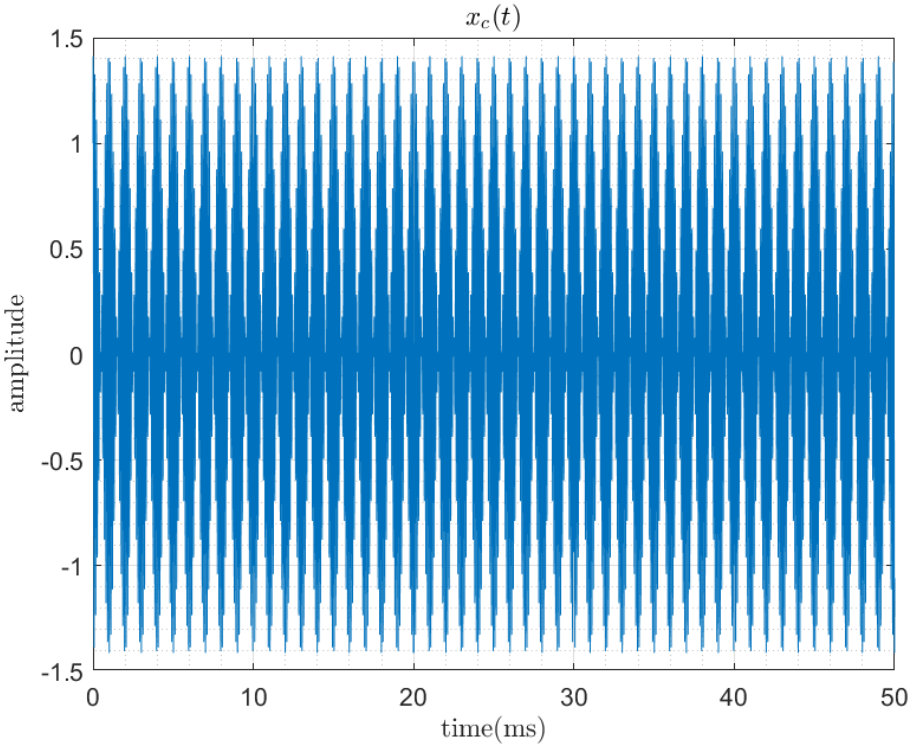


Figure 23: Modulated Signal - PSK

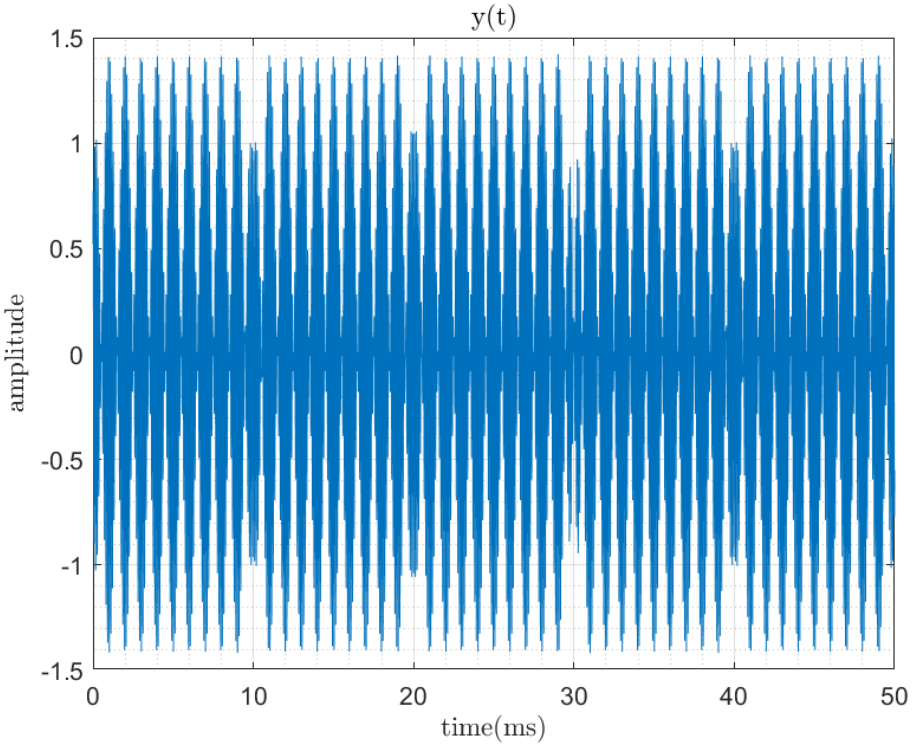


Figure 24: Channel‘s output - PSK

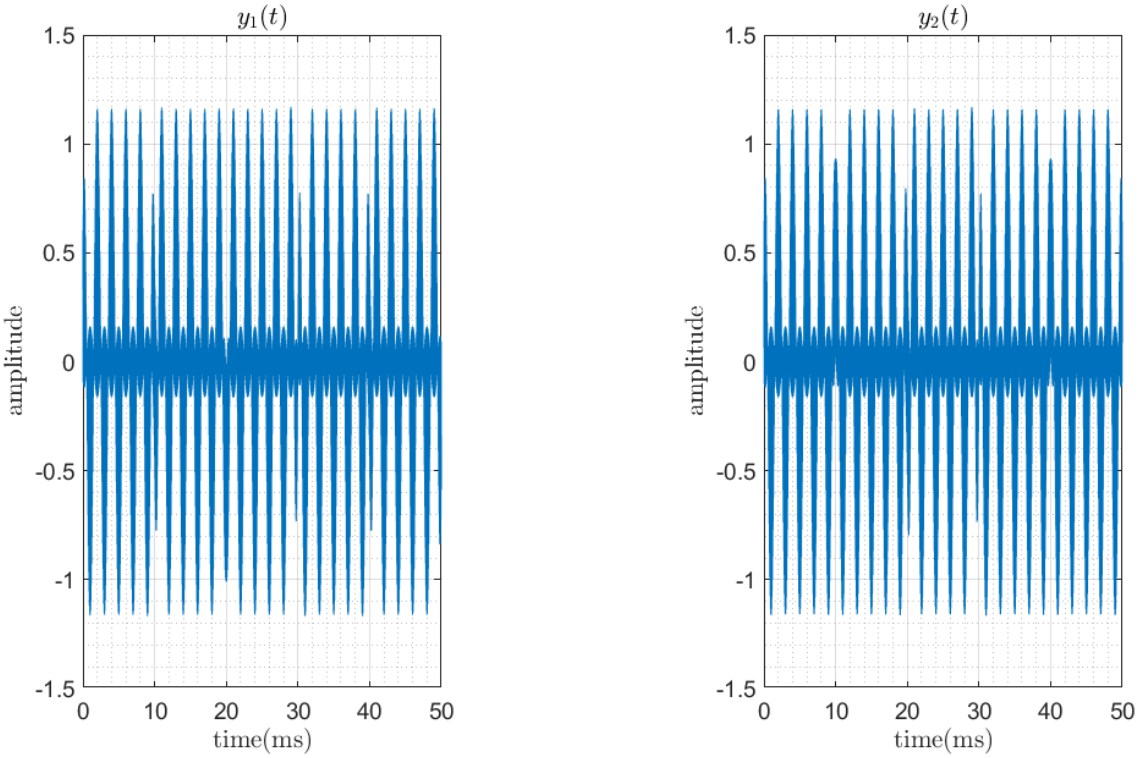


Figure 25: Demodulated Signals - PSK

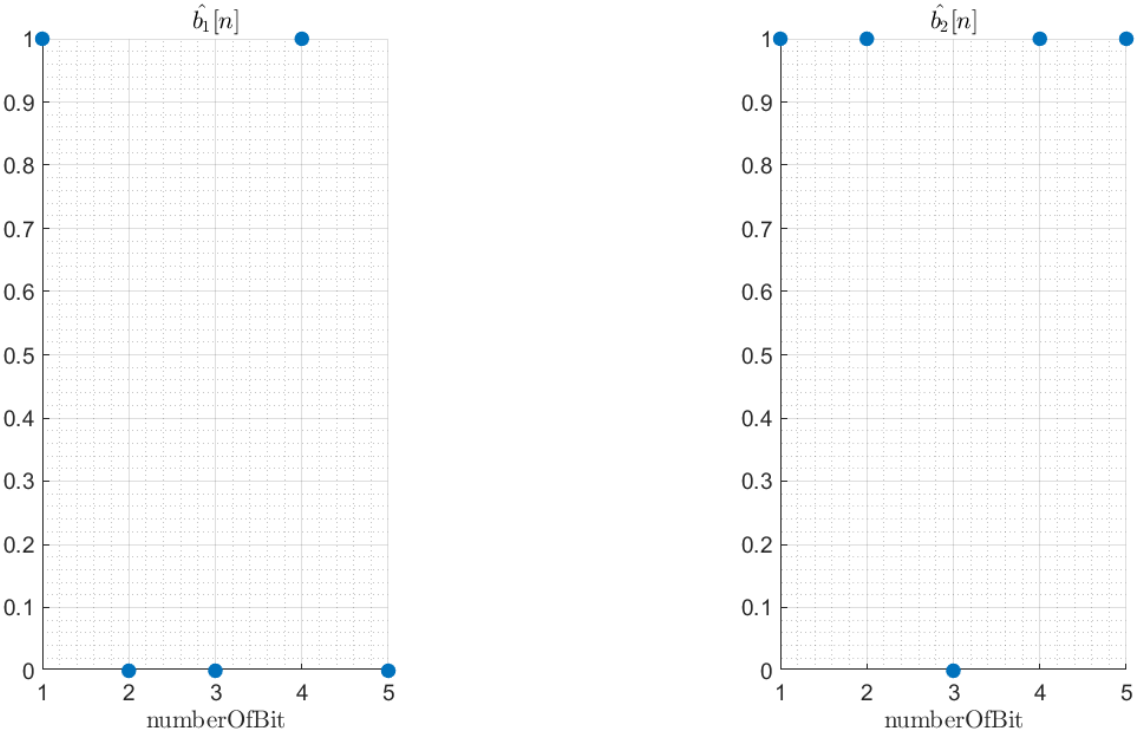


Figure 26: Outputs of Matched Filters - PSK

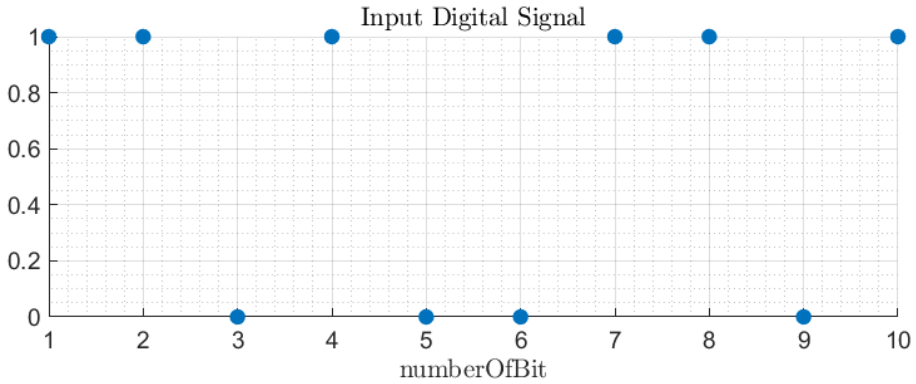


Figure 27: Input Signal - PSK

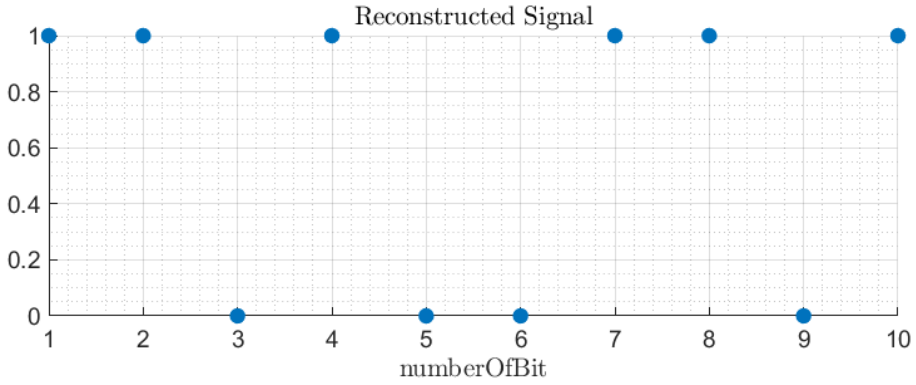


Figure 28: Reconstructed Signal - PSK

**Part.3.2.B - Adding Noise** As you can see, PSK seems to be more sensitive to noise since we have not zero probability of error at less SDs:

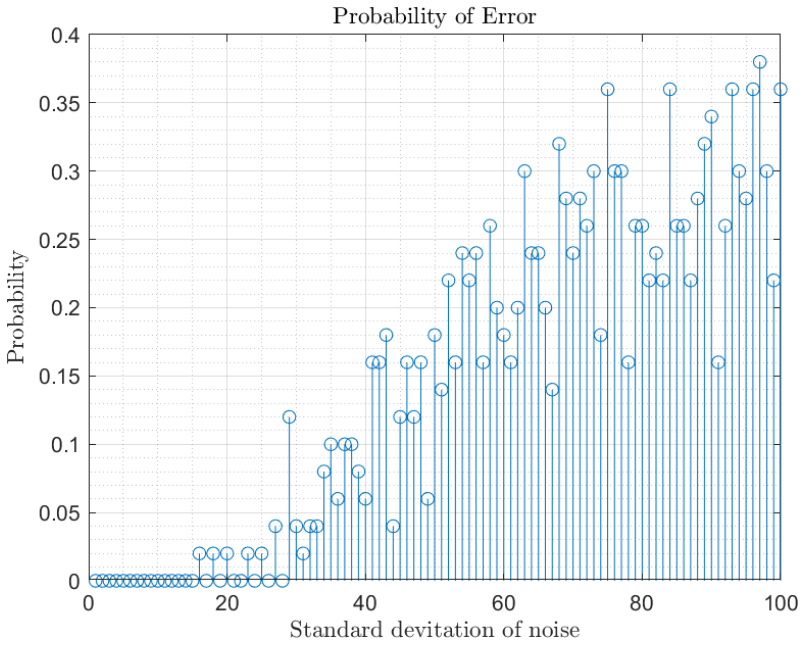


Figure 29: Error Probability - PSK

**Part.3.2.C - Signal Constellation** Using the same 6 SDs, [10 28 32 40 73 99], we plot the 2D Matched filter of the outputs of two matched filters:

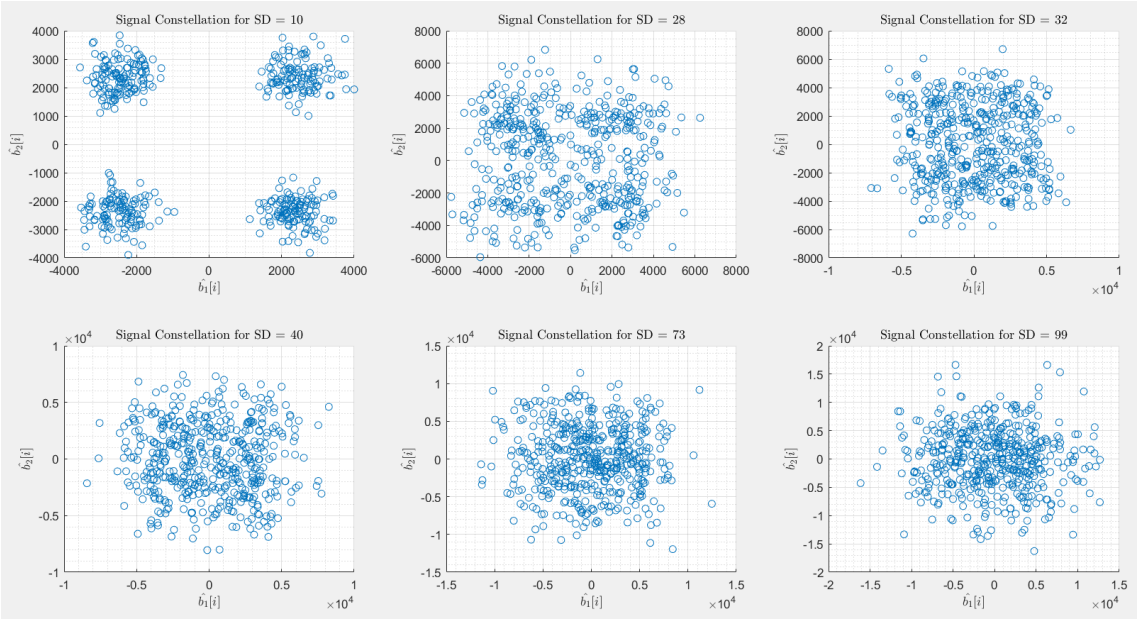


Figure 30: Signal Constellation of PSK

**Part.3.3 - FSK Modulation** Now we repeat all of the analysis in part.3.1 and 3.2 for FSK modulation, the only difference is in the p(t) format. Here p(t) is a cos(t) function with different frequencies depends on the bit sent:

```
// pulse
f1 = 1000; % in Hz
f0 = 1500; % in Hz
onePulse = cos(2*pi*f1*(0:1/Fs:tPulse-1/Fs));
zeroPulse = cos(2*pi*f0*(0:1/Fs:tPulse-1/Fs));
```

**Part.3.3.A** Yes! :

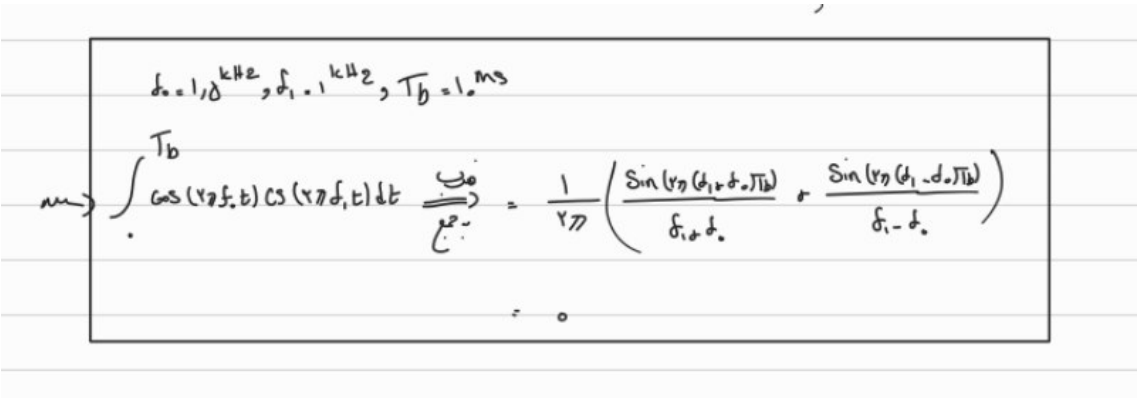

Figure 31: Orthogonal Signals

**Part.3.3.B - Simulation** The input signal with a length of 10: 1000101110 :

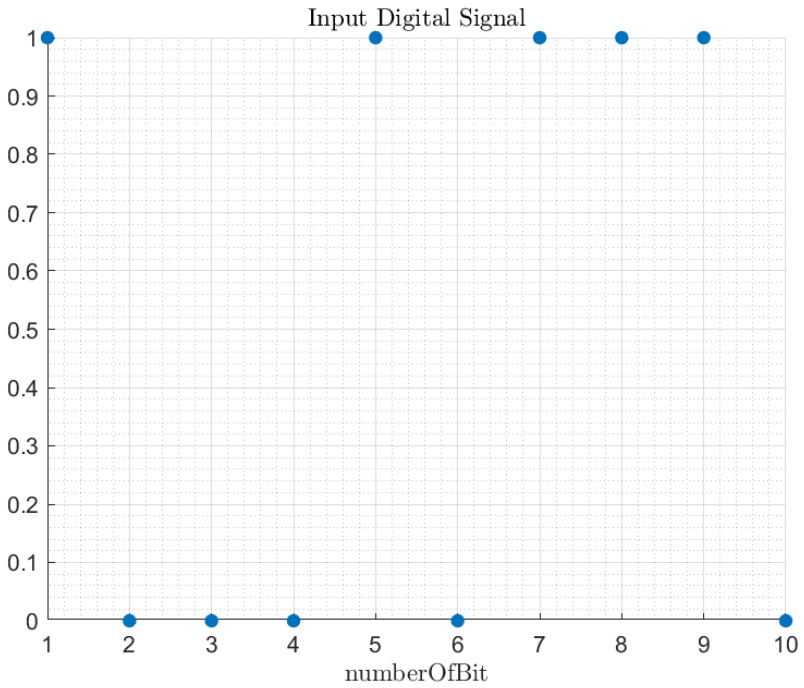


Figure 32: Input Digital Signal - FSK

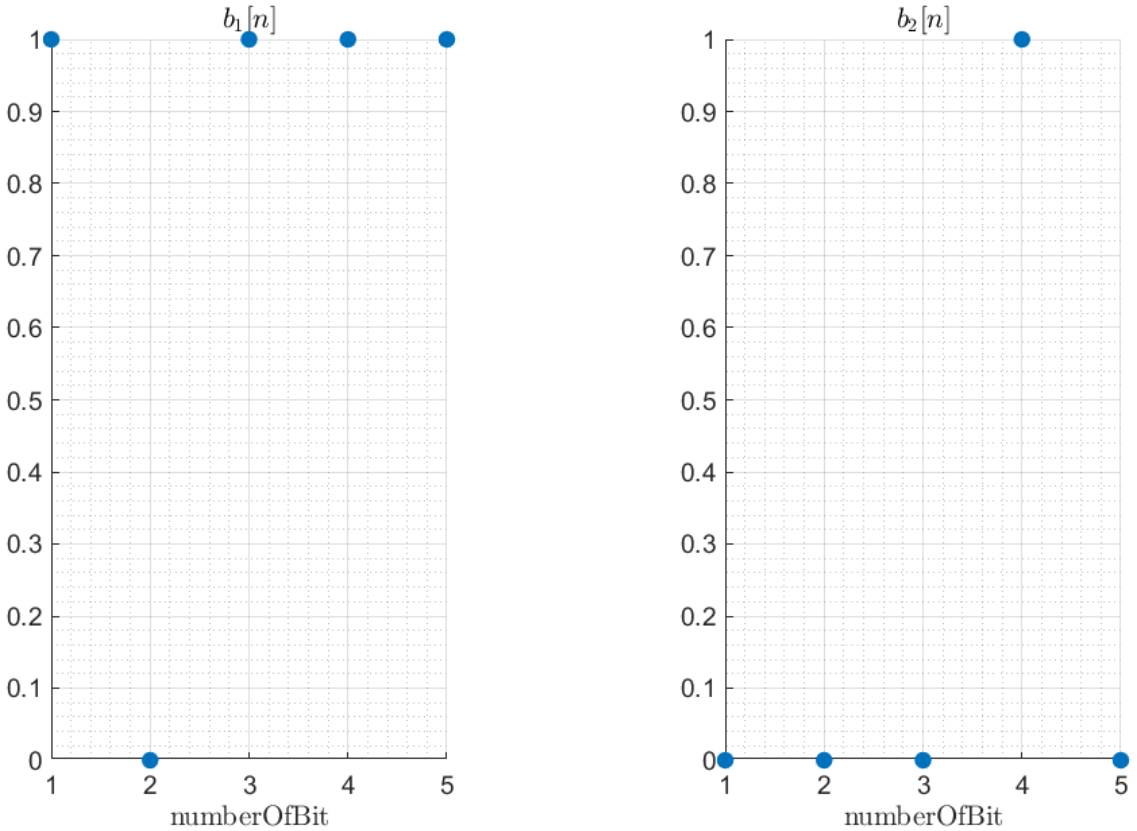


Figure 33: Divided Signals - FSK

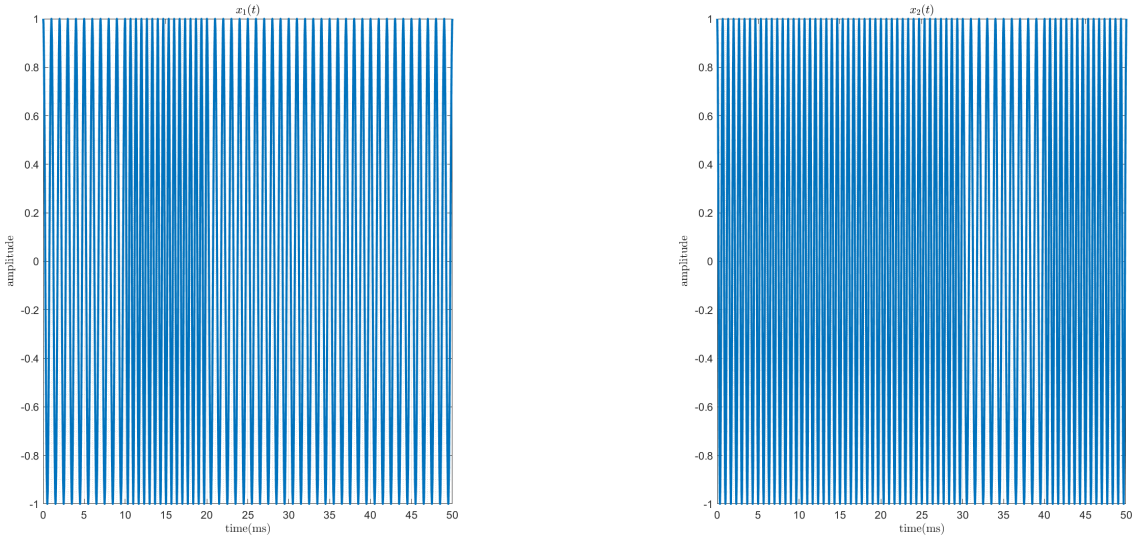


Figure 34: Analog Waveforms - FSK

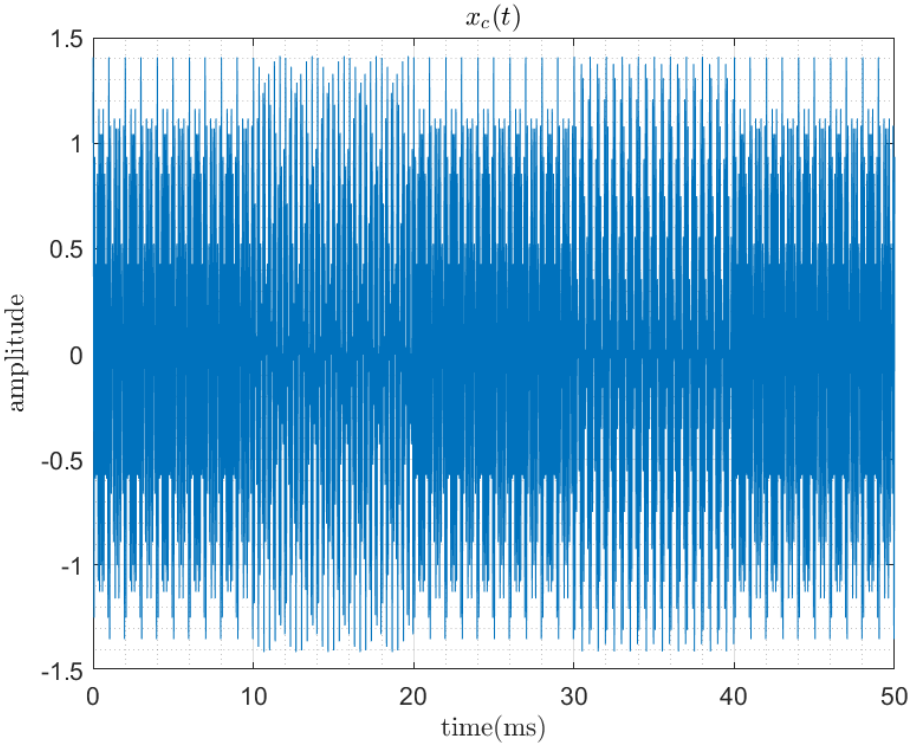


Figure 35: Modulated Signal - FSK

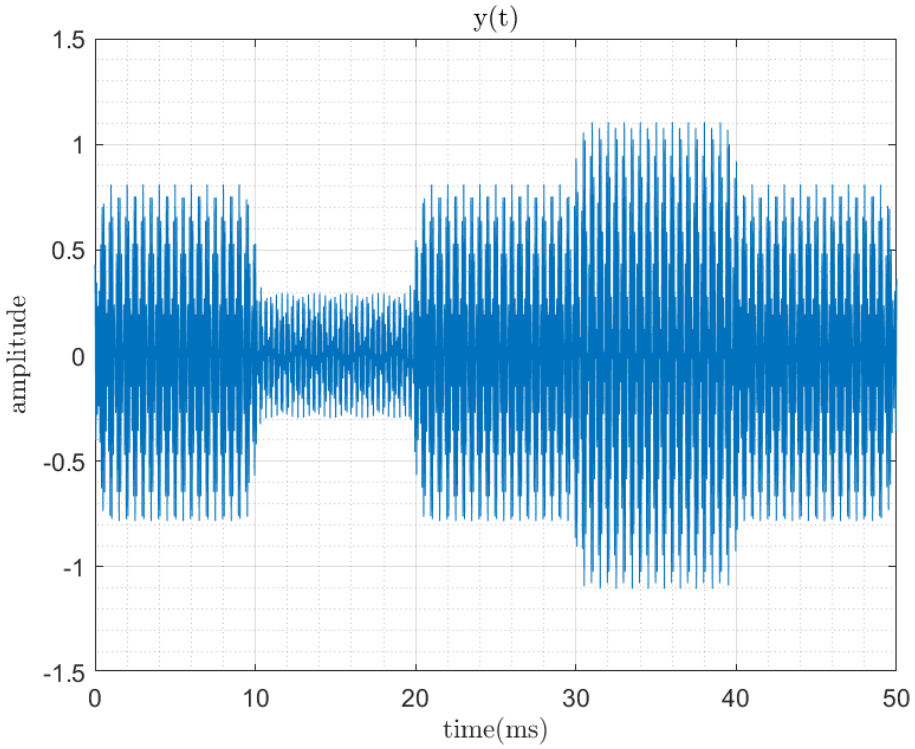


Figure 36: Channel's output - FSK

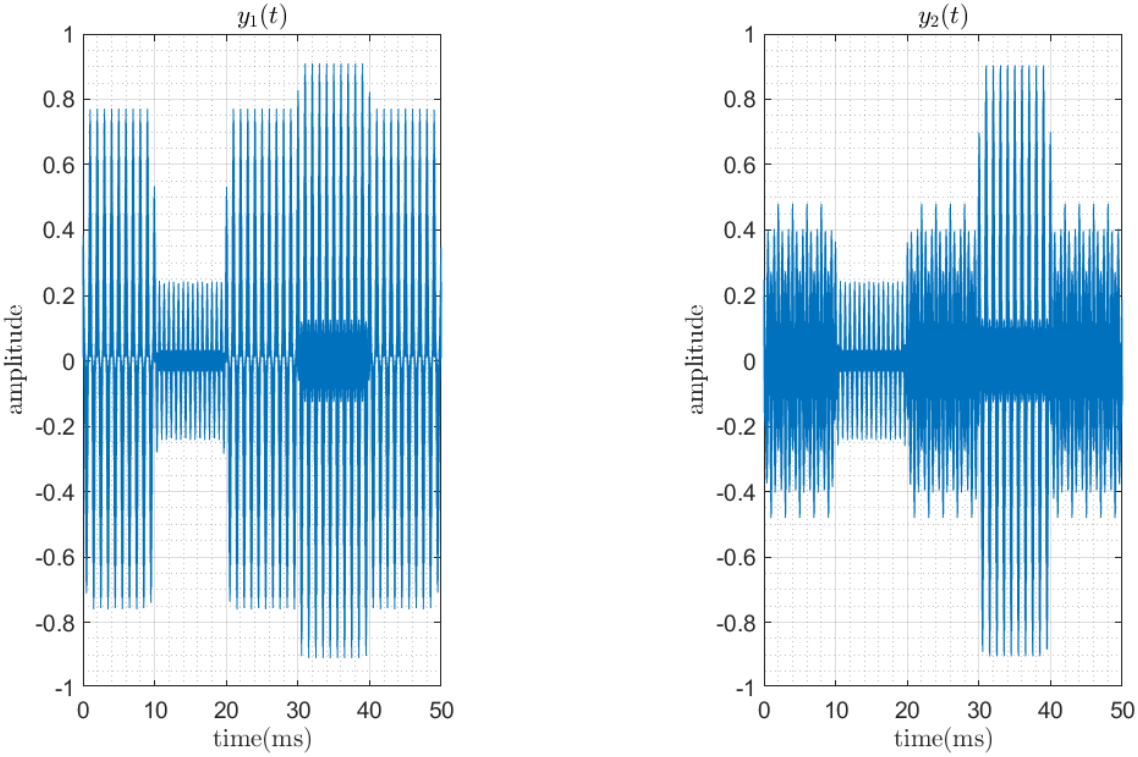


Figure 37: Demodulated Signals - FSK

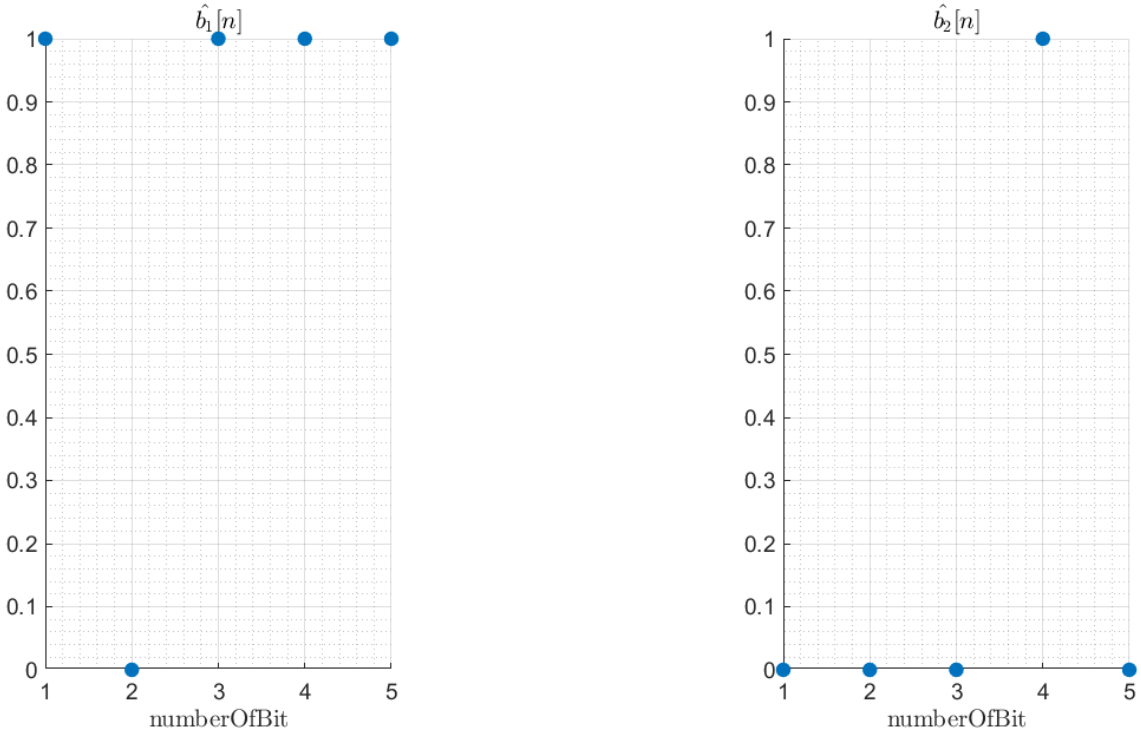


Figure 38: Outputs of Matched Filters - FSK

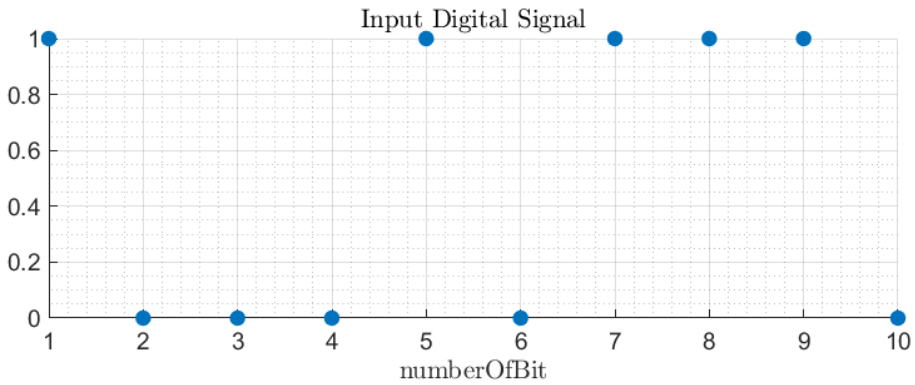


Figure 39: Input Signal - FSK

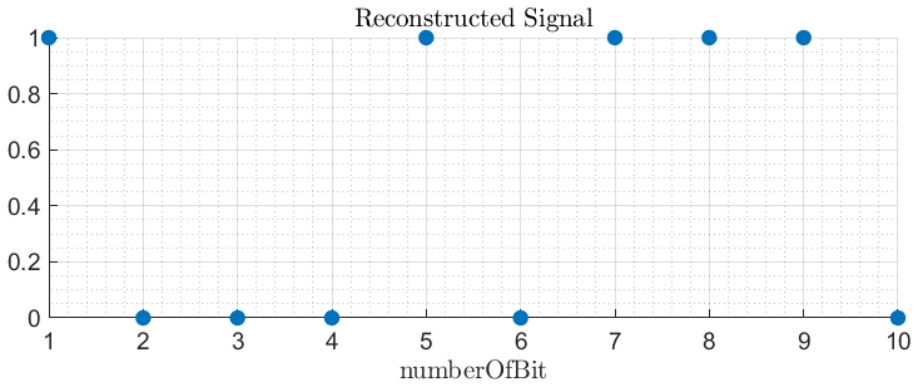


Figure 40: Reconstructed Signal - FSK

**Part.3.3.C - Adding Noise** As you can see, FSK seems to be the most sensitive to noise since we have not zero probability of error at less SDs:

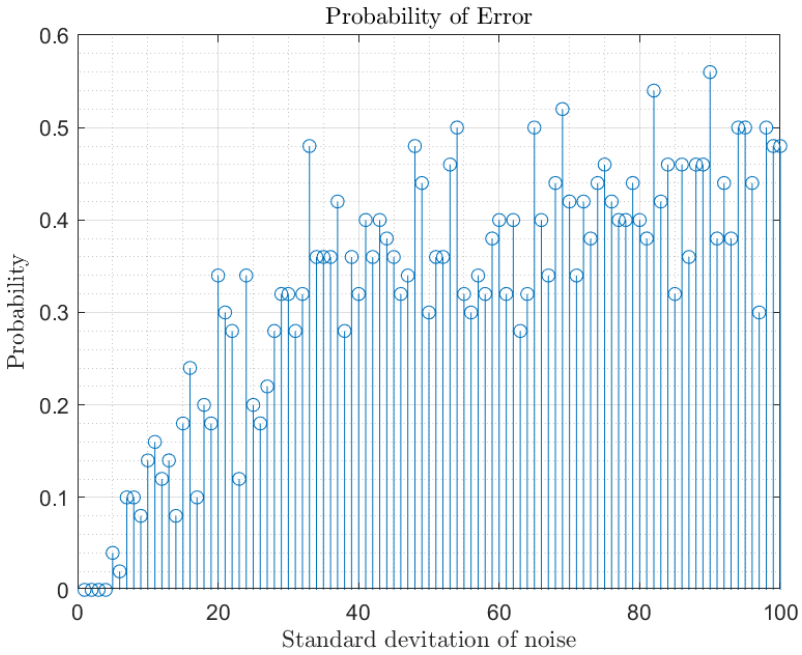


Figure 41: Error Probability - FSK

**Part.3.3.D - Signal Constellation** Using the same 6 SDs, [2 10 28 32 40 73], we plot the 2D Matched filter of the outputs of two matched filters:

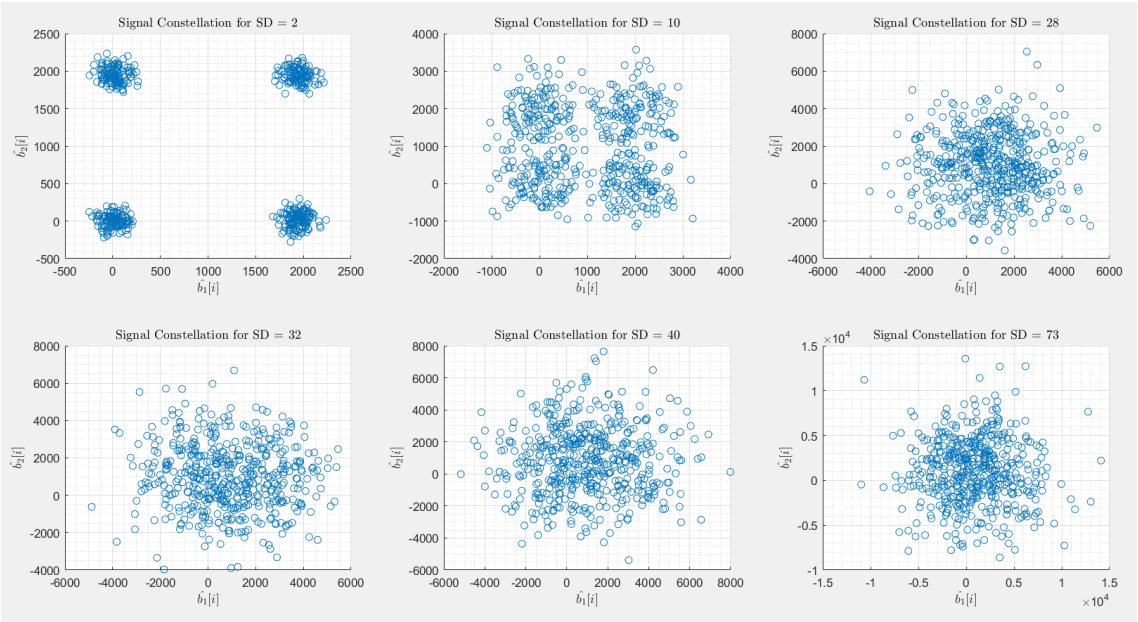


Figure 42: Signal Constellation of Fsk

**Part.3.4 - Conclusion** All the systems have 100% accuracy without noise but when the system is not ideal, According to the error probability plots, FSK is more sensitive to noise than PSK and PSK is more sensitive than PAM. It means, FSK and then PSK are start sensing noises with less variances. Also the probability of the error in FSK is more than PSK and then PAM. Signal constellation plots are supporting this statement, too. If we look at FSK signal constellation plots, at SD = 10 we have a lot of noise but for PSK and PAM for SD = 10, the points were focused around the 4 critical points.

# Part.4 - Transmitting a sequence of 8-bit numbers

**Part.4.1 - Source Generator/ Output Decoder** SourceGenerator function gets a sequence of decimal numbers in range of (0,255) and gives a sequence of binary numbers equals to that. OutputDecoder is the reverse of this function:

```
function output = SourceGenerator(inputSequence)
    output = de2bi(inputSequence);
end

function output = OutputDecoder(inputSequence)
    output = bi2de(inputSequence);
end
```

**Part.4.2 - Variance of Reconstruction Error** For each SD, we run the simulation 5 times and calculate the variance of the error values: (SD: 1:20:1000)

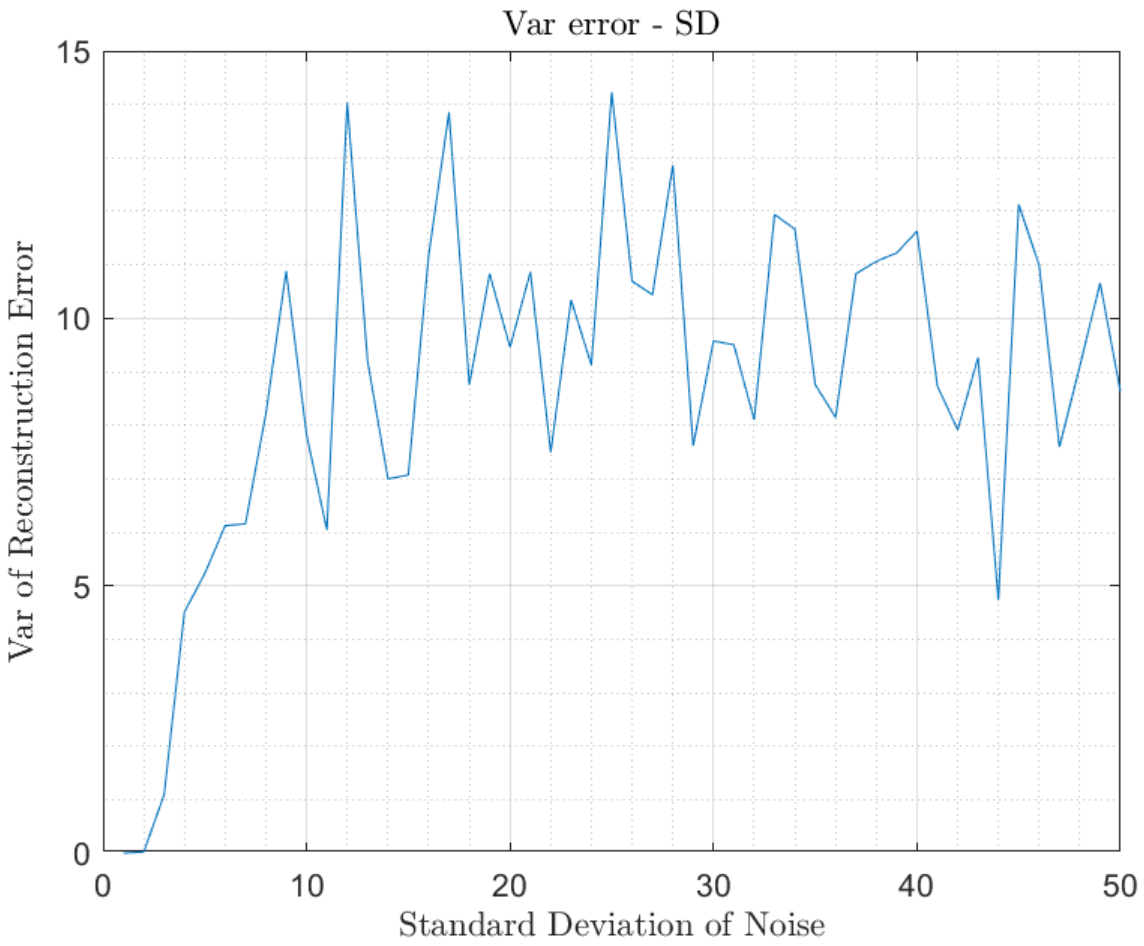


Figure 43: Var of Error vs SD of Noise

**Part.4.3 - Error Distribution** Error distribution for these SDs: [10 28 32 100 500 700]:

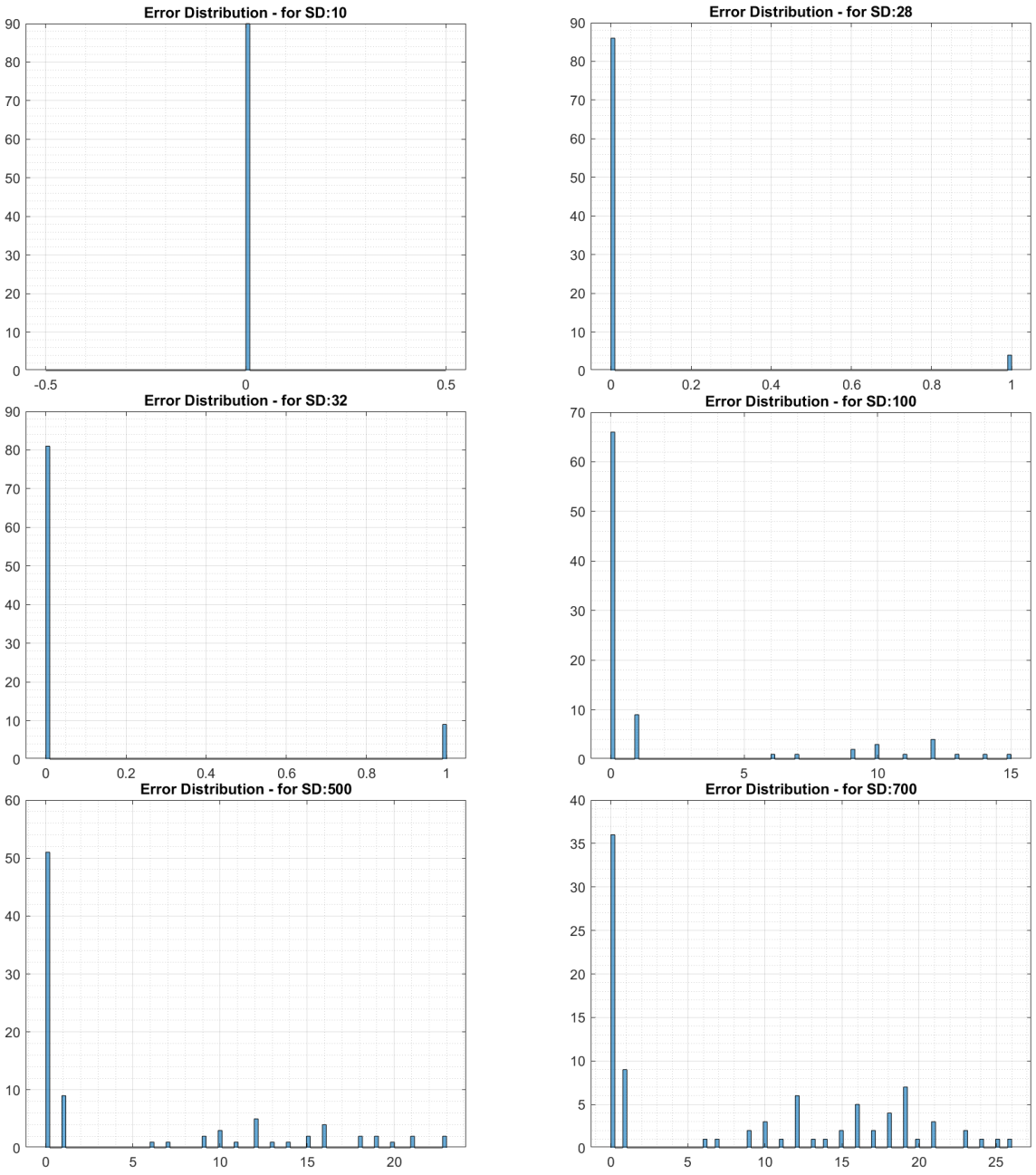


Figure 44: Error Distributions

# Part.5 - Source Coding

**Part.5.1** Codes of the question:

| $X$ | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| $C_1(X)$ | 0 | 10 | 11 | 10 | 01 | 111 |
| $C_2(X)$ | 0 | 10 | 01 | 010 | 11 | 111 |
| $C_3(X)$ | 0 | 01 | 011 | 0111 | 01111 | 011111 |

Figure 45: Code Words

a) $C_1(x)$ has two same codes which makes problem during coding! 10 is code for both b and d.
b) some of $C_2(x)$'s codes can be created by some others, for example, 010 could be 0+10 or 010 presenting a,b or d which this makes trouble during coding!
c) some of $C_3(x)$'s codes are at the first of other codes which we can't make decisions about the what is the incoming signal at time, for example if we see 0 we don't know it is "a" or we should wait for the next bit.

**Part.5.2** Solution of the optimization problem:

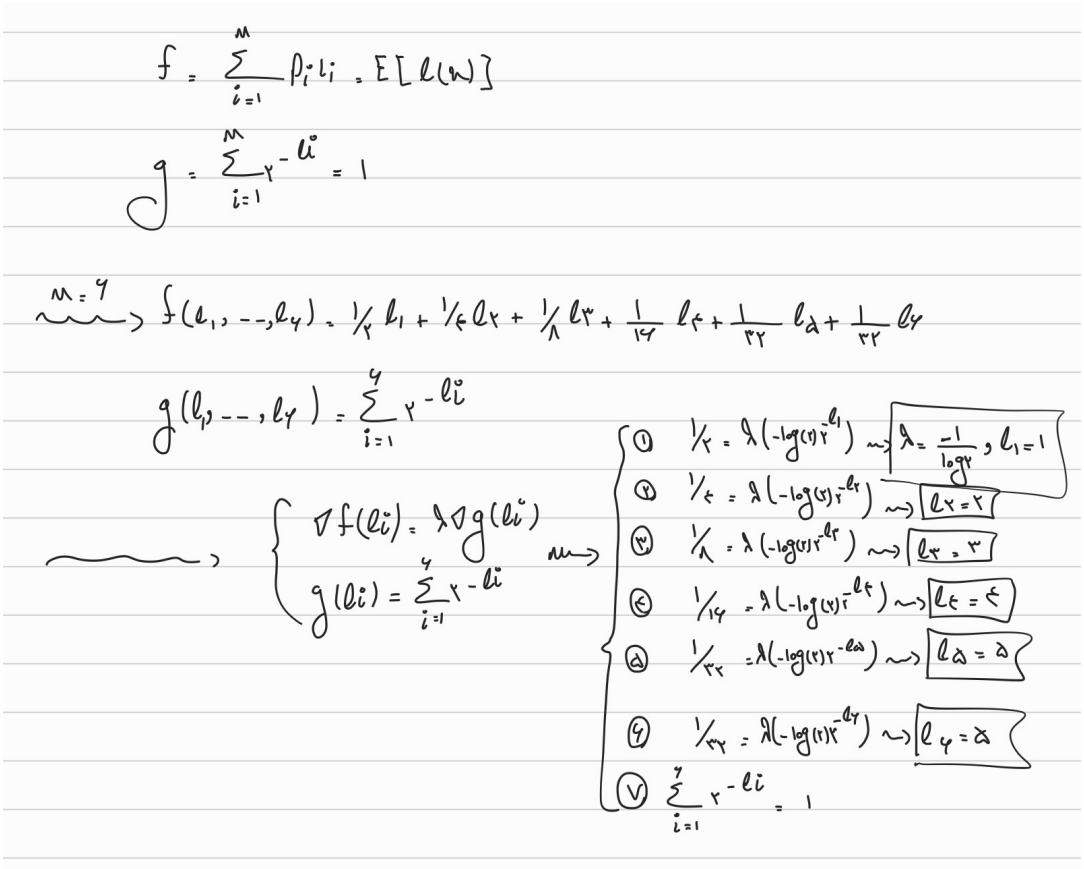

Figure 46: Length of code words

**Part.5.3** My code words:

| Symbol | Code |
|---|---|
| a | 1 |
| b | 01 |
| c | 001 |
| d | 0001 |
| e | 00001 |
| f | 00000 |

As we can see, these codes don‘t have any problems of $C_1(x)$, $C_2(x)$, $C_3(x)$.

**Part.5.4** Expectation Value of l(x):

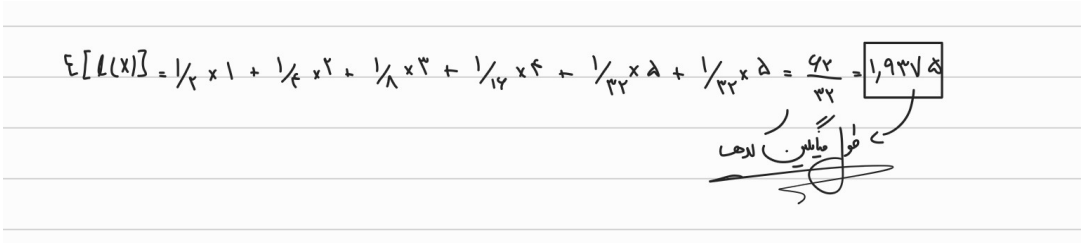

Figure 47: E[l(x)]

**Part.5.5** Using randsample, we create a vector of random symbols following their probabilities.

```
    //Information Source
    function output = InformationSource(n)
        symbols = ['a','b','c','d','e','f'];
        probabilities = [1/2,1/4,1/8,1/16,1/32,1/64];
        output = randsample(symbols,n, true, probabilities);
    end
```

**Part.5.6** We go trough all symbols and create their specified code in order!

```
    //Source Encoder
    function encodedSeq = SourceEncoder(n,seq)
    % codeWords after optimization problem
    % 1 - 01 - 001 - 0001 - 00001 - 00000
    % a - b - c - d - e - f
    codeWords = ['1', '01', '001', '0001', '00001', '00000'];
    encodedSeq = [];
    for i=1:n
        if (seq(i) == 'a')
            encodedSeq = [encodedSeq '1'];
        elseif (seq(i) == 'b')
            encodedSeq = [encodedSeq '01'];
        elseif (seq(i) == 'c')
            encodedSeq = [encodedSeq '001'];
        elseif (seq(i) == 'd')
            encodedSeq = [encodedSeq '0001'];
        elseif (seq(i) == 'e')
            encodedSeq = [encodedSeq '00001'];
        elseif (seq(i) == 'f')
            encodedSeq = [encodedSeq '00000'];
        end
    end
end
```

**Part.5.7 5.8** Going through all the sequence, we decode the code as implemented in the codes! An example of what these 3 functions do together:

```
sequence =

    'abbaabcbda'


encodedSeq =

    '1010111010010100011'


decodedSeq =

    'abbaabcbda'
```

Changing n from 1 to 5000, $H_n(x)$ will be:

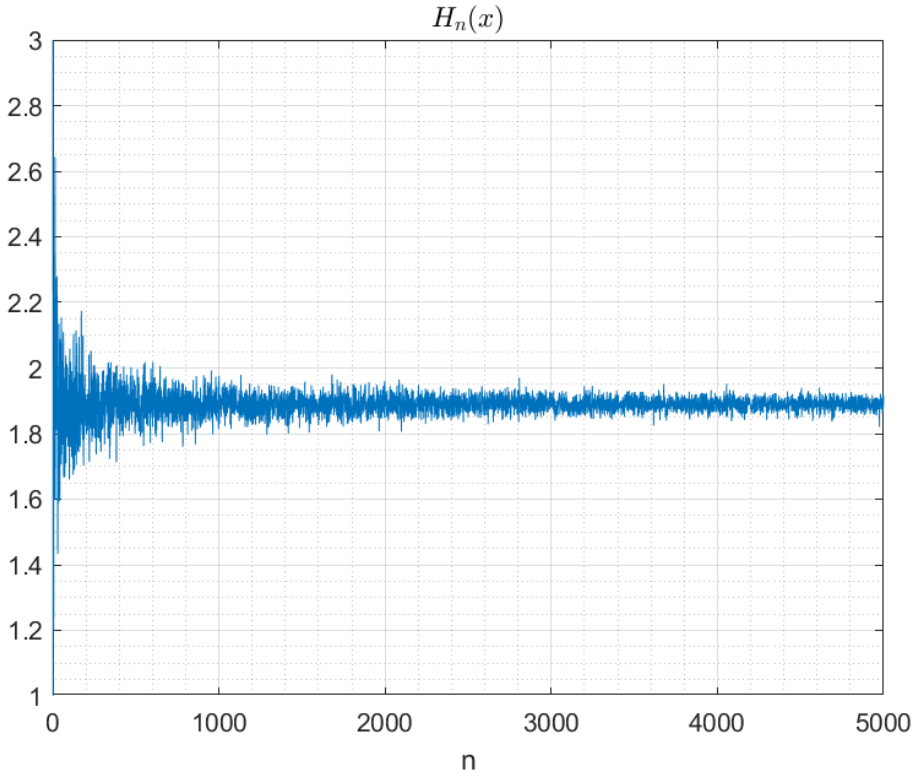


Figure 48: Averaged length of coded symbols

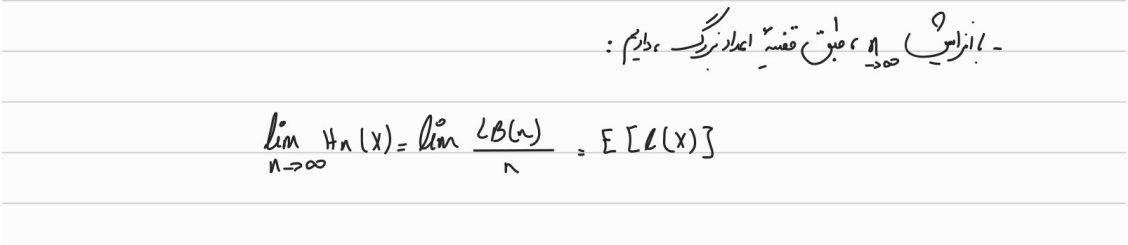


Figure 49: Theory