Communication Theory Project Report

Rohit Varma Chiluvuri
2022102028

Spring 2024

Contents

Abstract

This report documents the implementation and analysis of a BPSK communication system under memoryless and with-memory AWGN channels, as part of the Communication Theory course project. The effects of the channel characteristics on signal transmission and reception are examined, with plots and probability error analyses included to support observations.

1 Introduction

Binary Phase Shift Keying (BPSK) is a basic type of digital modulation that involves two distinct phase states of a carrier wave to represent binary values, 0 and 1. For each bit, which has a certain duration labeled as T, the carrier's phase is shifted to one of two positions that are 180 degrees apart. This arrangement of phases, where signals are exactly opposite, is called antipodal signaling.

1.1 Characteristics of BPSK

- **Simplicity:** BPSK is one of the simplest forms of phase modulation techniques, making it easy to implement and analyze.
- **Robustness:** The use of antipodal signals, where each signal phase is 180 degrees apart, provides good resistance to noise. This configuration ensures that the signal has the maximum possible distance between its states, thereby minimizing the probability of bit errors due to noise.
- **Efficiency:** Among binary modulation schemes, BPSK provides the lowest error probability for a given energy level. However, the error rate can be further reduced by using more complex modulation schemes that involve more than two signal states.

BPSK remains widely used in various communication systems due to these characteristics, offering a good balance between complexity and performance.

2 A/D Converter

2.1 Procedure

2.1.1 Code

Listing 1: Analog to Digital

```
[~, fs] = audioread('project.wav');
wavdata = [120/128, 250/128, 10/128, 115/128];
%% SNR Configuration
snr = 10;
%% SNR Output
out = zeros(1, length(snr));
%% Main SNR loop
```

```
for z = 1:length(snr)
    %% Quantization
    range = 2;
    level8 = range / (2^8);
    eight = round(wavdata / level8) * level8;
    Tb = 1/(fs*9);
    %% A/D Conversion
    scaled_in = round(eight * 128);
    ad_out = zeros(length(scaled_in) *9,1);

    for i = 1:length(scaled_in)
        binStr = dec2bin(scaled_in(i), 9);
        binArray = binStr - '0';
        ad_out((i-1)*9 + 1:i*9) = binArray;
    end
    encoded = ad_out;
```

2.1.2 Implementation Details

- **Audio File Reading:** The function `audioread` is used to read the audio file 'project.wav'. It returns the sampling frequency fs, which determines the number of samples per second in the audio file. In this script, the actual data from `audioread` is not used; instead, a predefined array of normalized values is manually specified.
- **Quantization:** The signal is quantized to 256 levels by dividing a predefined range (2) by 2^8. Each value from the input data is then rounded to the nearest level, This is a technique in digital audio processing to reduce the data rate or the amount of data needed to store the signal.
- **Analog-to-Digital Conversion:** Each quantized value is scaled up by 128 (assuming the input is between 0-2 after adding one to the wavdata)and then converted into a binary string of 9 bits. These binary strings are then converted into arrays of binary digits, which are sequentially stored in the output array `ad_out`. This step simulates the conversion of analog signals into digital form, which is essential for digital signal processing and storage.
- **Binary Representation:** Each quantised value of the original audio is represented by a 9-bit binary number , allowing for digital processing and transmission.

2.2 Plots

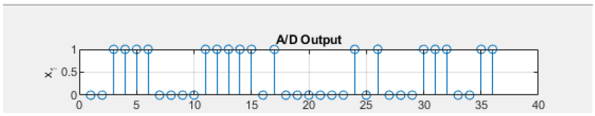

Figure 1: A/D output plot

We successfully convert the four test samples into 36 bits ready for encoding.

3 Encoder

We enCode the output of the A/D converter using BPSK Map. In BPSK, each bit of the digital data is mapped to a phase value, forming the simplest form of phase shift keying modulation where each bit is represented by one of two phases.

3.1 BPSK Mapping

In Binary Phase Shift Keying (BPSK), the binary values (0 and 1) from the output of the A/D conversion are mapped to a signal phase of 180° difference. The mapping is typically defined as follows:

- A binary '1' is mapped to $\cos(2\pi ft)$, representing a carrier wave with zero phase shift.
- A binary '0' is mapped to $-\cos(2\pi ft)$, representing a carrier wave with a phase shift of π (180 degrees).

This phase shift keying allows the representation of the digital binary data as a physical waveform that can be transmitted over communication channels.

3.1.1 Mathematical Representation

The BPSK signal $s(t)$ can be mathematically represented as:

$$s(t) = A \cdot (-1)^b \cdot \cos(2\pi f_c t)$$

where:

- A is the amplitude of the carrier signal.
- b is the binary data bit (0 or 1).
- f_c is the frequency of the carrier.

The modulation involves a change in the phase of the carrier signal according to the binary data bit, where a '0' induces a phase shift of π and a '1' results in no phase shift.

3.2 Procedure

Explanation of how the encoding is performed and choices made regarding parameters.

3.2.1 Code

Listing 2: Encoding

```
for i = 1:length(encoded)
    if(encoded(i) == 0)
        encoded(i) = -1;
    end
end
```

3.2.2 Implementation Details

The conversion process is implemented using a simple conditional loop in MATLAB This loop iterates through each element of the array `encoded`, which contains binary values. If a value is '0', it is converted to '-1'. This manipulation transforms the binary data into a form where '1' and '-1' directly correspond to the phase shifts of '0' and 'π' radians, respectively, which is characteristic of BPSK modulation.

3.3 Plots

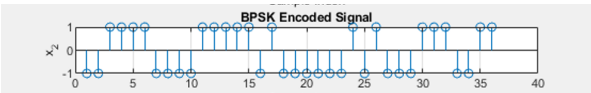

Figure 2: BPSK encoded Plot

We successfully encoded the bits using BPSK mapping.

4 Line Coding

4.1 Procedure

4.1.1 Code for Raised Cosine:

Listing 3: Line Coding

```
    encoded_upsample = upsample(encoded,17);
    a = 1;
    m = 9;
    len = 2;
    [rc, time] = raised_cosine(a, m, len);
    rc = rc.*(1/max(rc));
    encoded_upsample = upsample(encoded, length(rc));

function [rc,time_axis] = raised_cosine(a,m,len)
    len_os = floor(len*m);
    z = cumsum(ones(len_os,1))/m;
    A= sin(pi*(1-a)*z)./z;
    B= cos(pi*(1+a)*z)*4*a;
    C= (1 - (4*a*z).^2)*pi;
    zerotest = m/(4*a);
    if (zerotest == floor(zerotest))
    B(zerotest) = cos(pi*(1+a)*(z(zerotest)+0.001))*4*a;
    A(zerotest) = sin(pi*(1-a)*(z(zerotest)+0.001))./(z(zerotest)+0.001);
    C(zerotest) = (1-(4*a*(z(zerotest)+0.001)).^2)*pi;
    end
    D = (A+B)./C;
    rc = [flipud(D);1;D];
    rc(len_os+1)=(4*a/pi) + (1-a);
```

```
    time_axis = [flipud(-z);0;z];
end
```

4.1.2 Code for Rectangular pulse:

Listing 4: Line Coding

```
  for i = 1:length(rc)
      if (i >= 1 && i <= 37)
          rc(i) = 1;
      else
          rc(i) = 0;
      end
   end
```

4.1.3 Implementation details for Raised Cosine:

The Raised Cosine filter is implemented to shape the line coding of the encoded signals. Key steps in the implementation include:

- **Upsampling**: The encoded data is upsampled by a factor of 17 to prepare for pulse shaping, which increases the sample rate, making room for the shape of the pulse to be inserted between the original data points.
- **Filter Design**: Parameters for the Raised Cosine filter include:
  - Roll-off factor (`a`) set to 1, which determines the steepness of the roll-off.
  - Samples per symbol (`m`) set to 9.
  - Length (`len`) set to 2, affecting the overall duration of the pulse.
- **Pulse Calculation**: The actual Raised Cosine pulse is calculated using a custom function that creates the pulse based on the given parameters. The function computes the time-domain representation of the pulse, ensuring that it adheres to the desired spectral properties.
- **Normalization**: The pulse is normalized to ensure that its maximum value is 1, maintaining consistent amplitude across the signal.

4.1.4 Implementation details for Rectangular Pulse:

The implementation of a simple Rectangular pulse involves defining the pulse shape directly within the sample stream:

- **Pulse Definition**: A loop iterates through the length of the Raised Cosine array (`rc`), setting the first 37 samples to 1 and the remainder to 0.
- **Application**: This creates a sharp pulse shape which is typically used in simple digital communications systems where bandwidth efficiency is not a primary concern.

4.2 Plots

4.2.1 Rectangular Pulse

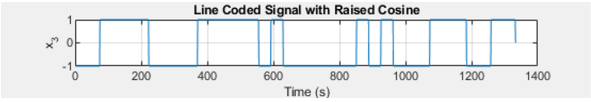

Figure 3: Line Code with Rectangular Pulse Transmit filter

4.2.2 Raised Cosine

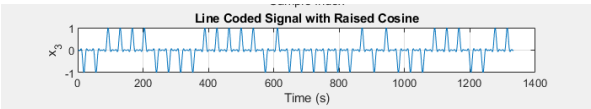

Figure 4: Line Code with Raised Cosine Transmit filter

We can see how samples are line coded, where samples are convoloted with filter to get line coded signal.

5 Modulation

5.1 Procedure

5.1.1 Code

Listing 5: Modulation label

```
%% Line Output
out_line = conv(rc, encoded_upsample);
out_line = out_line(1:(length(out_line) - (length(rc) - 2)));

%% Modulation
fc = 1e6;
t = 0:1/(10*fc):(length(out_line) - 1)/(10*fc);
modulated = out_line.*cos(2*pi*fc*t)';
```

5.1.2 Implementation details

The implementation involves two primary steps:

- **Line Output Creation**: The digital signal is shaped by convolving the encoded and upsampled signal with the selected pulse shape. The resulting signal is truncated to match the original length by removing excess samples from the convolution process.
- **Signal Modulation**: The shaped signal is modulated using a sinusoidal carrier wave. The modulation process multiplies the line output by a cosine function with a frequency of 1 MHz, effectively shifting the signal's spectrum for transmission.

5.2 Plots

5.2.1 Rectanglular Pulse

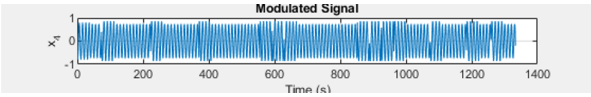

Figure 5: Modulation with rectangular pulse

5.2.2 Raised cosine Pulse

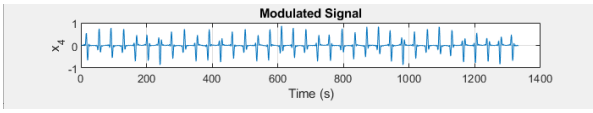

Figure 6: Modulation with Raised Cosine pulse

6 Channel

More details about channel are provided in further sections.

6.1 Plots

6.1.1 Rectangle pulse

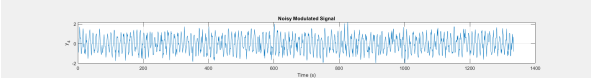

Figure 7: Modulated signal after noise with rectangular pulse and SNR = 10

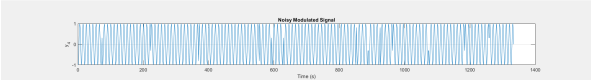

Figure 8: Modulated signal without noise with rectangular pulse

6.1.2 Raised cosine pulse

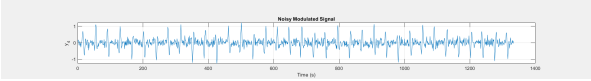

Figure 9: Modulated signal with noise with raised cosine pulse and SNR = 10

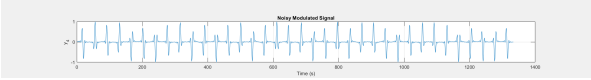

Figure 10: Modulated signal without noise with raised cosine pulse

6.2 AWGN Channel with Memory

6.3 Implementation

6.4 Plots

6.4.1 Rectangle pulse

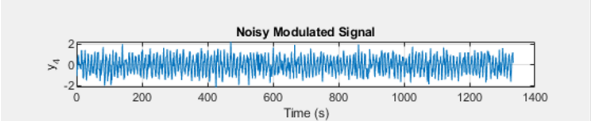

Figure 11: plot with SNR 10 with rect pulse:

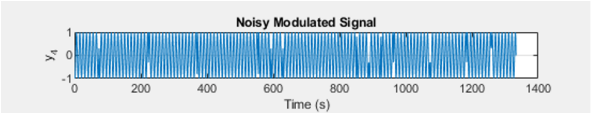

Figure 12: Plot with no noise with rect pulse:

6.4.2 Raised cosine pulse

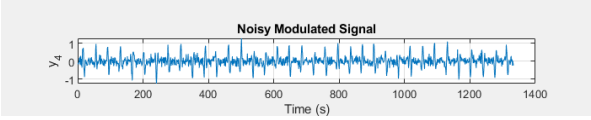

Figure 13: plot with SNR 10 with raised cosine pulse:

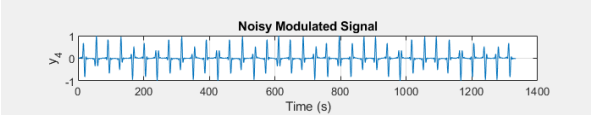

Figure 14: plot with no noise with raised cosine pulse:

7 Demodulation

7.1 Procedure

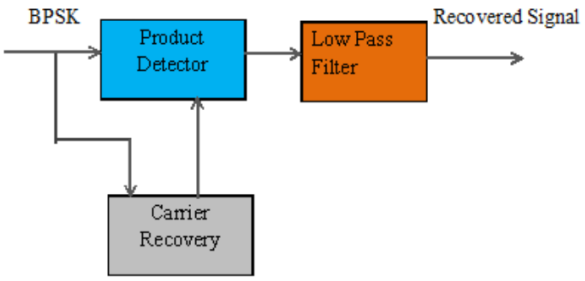

Figure 15: Demodulation Block Diagram

1. The incoming BPSK signal is fed into a product detector. Here, it is multiplied by a locally generated carrier signal that has the same frequency as the original carrier. This step is crucial for converting the frequency of the received signal back to baseband.

2. This block is responsible for generating a carrier signal that is synchronized with the phase and frequency of the received BPSK signal. Accurate carrier recovery is vital for the successful demodulation of the BPSK signal.

3. After the product detection, the signal contains the desired baseband information along with high-frequency components due to the carrier doubling. The low pass filter removes these high frequencies, leaving only the baseband binary signal.

4. The output of the low pass filter is the recovered line coded signal, which should ideally match the original signal transmitted after being line encoded into BPSK.

This demodulation scheme effectively reconstructs the original digital data by reversing the modulation process used in transmission.

7.1.1 Code

Listing 6: Demodulation

```
demod = 2*cos(2*pi*fc*t);
demodulated_blp = noisy_modulated.*demod';
demodulated_lp = lowpass(demodulated_blp, cutoff, fs);
```

7.2 Plots:

7.2.1 Rectangle Pulse

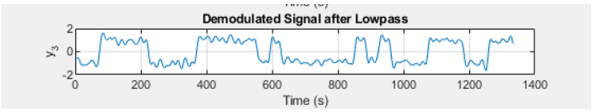

Figure 16: plot with SNR 10 with rect pulse:

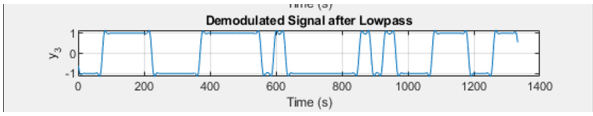

Figure 17: plot with no noise with rect pulse:

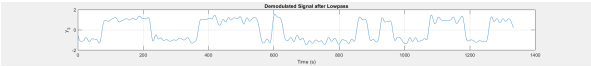

Figure 18: plot with SNR 10 with rect pulse(Memoryless noise:)

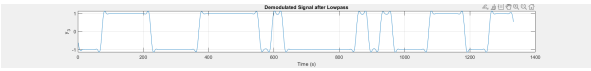

Figure 19: plot with no noise with rect pulse(Memoryless noise:)

7.2.2 Raised Cosine pulse:

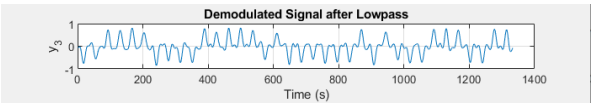

Figure 20: plot with SNR 10 with raised cosine pulse

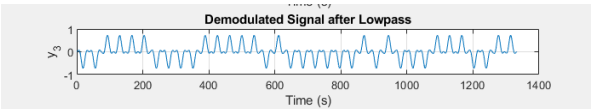

Figure 21: plot with no noise with raised cosine pulse

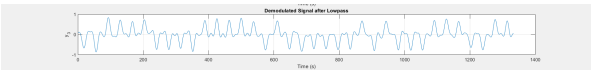

Figure 22: plot with SNR 10 with raised cosine pulse(Memoryless noise

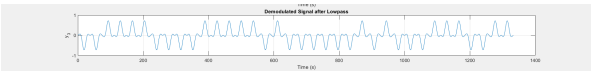

Figure 23: plot with no noise with raised cosine pulse(memoryless noise)

8 Line Decode

8.1 Procedure

Line decode makes the decision for the output statistics.

8.1.1 Code

Listing 7: Line Decoding

```
demodulated_lp_cluster = reshape(demodulated_lp(1:end-1),17,[]);

line_decoded = zeros(size(demodulated_lp_cluster, 2), 1);

for i = 1:size(demodulated_lp_cluster, 2)
    row = demodulated_lp_cluster(:, i);
    [maximum, idx] = max(abs(row));

    line_decoded(i) = row(idx);
end
subplot(3,1,2);
stem(line_decoded);
```

8.2 Plots

8.2.1 Rectangular pulse

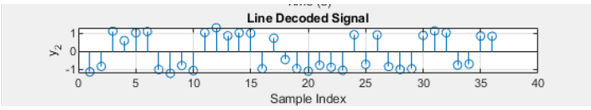

Figure 24: plot with SNR 10 with rect pulse

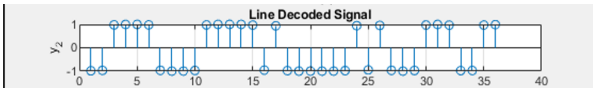

Figure 25: plot with no noise with rect pulse

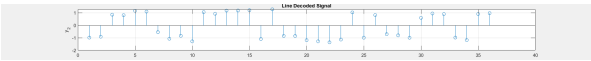

Figure 26: plot with SNR 10 with rect pulse(memoryless noise)

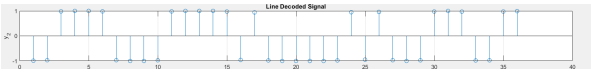

Figure 27: plot with no noise with rect pulse(memoryless noise)

8.2.2 Raised Cosine pulse

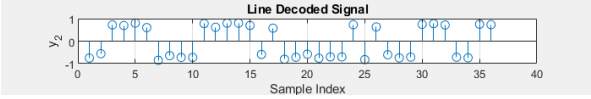

Figure 28: plot with SNR 10 with raised cosine pulse

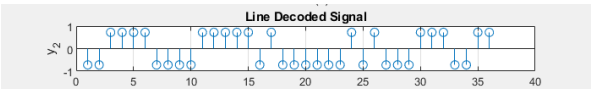

Figure 29: plot with no noise with raised cosine pulse

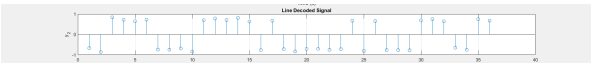

Figure 30: plot with SNR 10 with raised cosine pulse(memoryless noise)

Figure 31: plot with no noise with raised cosine pulse(memoryless noise

9 Decoder

This decodes the decision statistics to give the final decoded bits to be converted back to analog.

9.1 Procedure

In BPSK demodulation, a basic decision rule is applied to determine the binary value from the received signal. The rule can be described succinctly as follows:

- If the output is greater than 0, it is mapped to binary '1'.
- If the output is less than or equal to 0, it is mapped to binary '-1'.

9.1.1 Code

Listing 8: Decoder

```
decoded = zeros(1,length(line_decoded));
for i = 1:length(decoded)
    if(line_decoded(i)>0)
        decoded(i) = 1;
    else
        decoded(i) = 0;
    end
end

subplot(3,1,3);
stem(decoded);
```

9.2 Plots:

9.2.1 Rectangle pulse

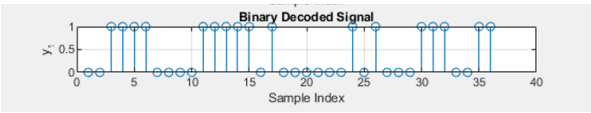

Figure 32: plot with SNR 10 with rect pulse

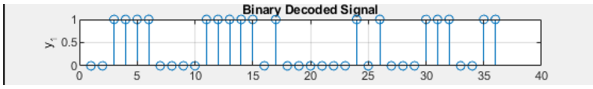

Figure 33: plot with no noise with rect pulse

Figure 34: plot with SNR 10 with rect pulse(memoryless noise)

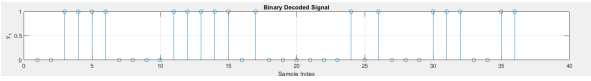

Figure 35: plot with no noise with rect pulse(memoryless noise)

9.2.2 Raised Cosine pulse

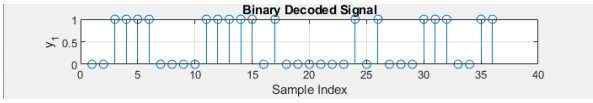

Figure 36: plot with SNR 10 with raised cosine pulse

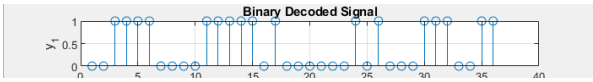

Figure 37: plot with no noise with raised cosine pulse

Figure 38: plot with SNR 10 with raised cosine pulse(memoryless noise)

Figure 39: plot with no noise with raised cosine pulse(memoryless noise)

10 D/A

10.1 Procedure

This section of the code handles the conversion of the decoded binary data back into analog format. The process involves several key steps:

- **Reshaping:** The `decoded` array, which contains binary data, is reshaped into a matrix with 9 rows. This reshaping prepares the binary data for grouping into 9-bit words, which correspond to the format used during the encoding process.
- **Initialization:** An output array `da_out` is initialized with zeros. The length of this array is set to the number of columns in the reshaped `decoded_temp` matrix, representing the number of 9-bit words.
- **Conversion Loop:**
  - Each column of the reshaped matrix `decoded_temp` is converted to a string of binary numbers without spaces using the `num2str` function followed by `strrep` to remove any spaces.
  - The binary string is then converted back to a decimal number using `bin2dec`, which interprets the string of '0's and '1's as a binary number.
  - This decimal number is scaled down by 128 (normalizing it back to its original range) and stored in the `da_out` array.

This method effectively converts the digital data, which was modified for transmission as a binary stream, back into a format resembling its original analog form.

10.1.1 Code

Listing 9: Digital to Analog

```
decoded_temp = reshape(decoded, 9, []);
da_out = zeros(1, size(decoded_temp, 2));
for i = 1:size(decoded_temp, 2)
    binary_string = num2str(decoded_temp(:, i)');
    binary_string = strrep(binary_string, ' ', '');
    da_out(i) = bin2dec(binary_string) / 128;
end
```

11 Final Plot

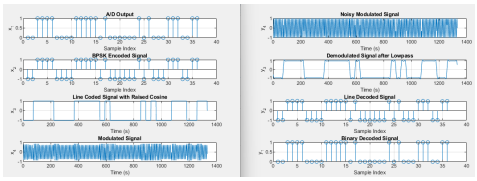

Figure 40: Output plots of each block

12 Comparative Analysis of AWGN Channels

12.1 Memoryless AWGN Channel

Characteristics:

- The memoryless AWGN channel introduces white Gaussian noise with a constant spectral density at each instant, independent of other times.
- The received signal $r(t) = s(t) + n(t)$ where $n(t)$ represents the noise component.

PDF and Impact on BPSK:

- Noise $n(t)$ is Gaussian distributed with zero mean and a variance σ^2.
- BPSK uses binary values where '0' is mapped to $-a$ and '1' to $+a$. The decision boundary is at 0, making the probability of error (Pe) calculable via the Q-function.

12.2 AWGN Channel with Memory

Characteristics:

- This channel type exhibits memory by introducing dependencies in noise across different times via convolution $r(t) = h(t) * s(t) + n(t)$.

- The impulse response $h(t) = a\delta(t) + (1-a)\delta(t-bT_b)$ shows that noise affects the signal at current and specific past times.

PDF and Impact on BPSK:

- While the noise component remains Gaussian, the convolution introduces correlations, complicating the noise distribution and making it effectively non-Gaussian.
- Demodulation processes need to account for these correlations, often requiring advanced techniques like equalization.

12.3 Comparative Analysis

Key Differences:

- **Noise Influence:** Memoryless channels treat each symbol independently concerning noise, whereas channels with memory introduce correlations that can affect symbol integrity.
- **Probability of Error:** Pe is generally lower in memoryless channels under similar conditions due to simpler noise characteristics. In channels with memory, correlated noise tends to increase Pe.
- **Receiver Complexity:** Receivers for memoryless channels primarily need to handle Gaussian noise, which is less complex than those required for channels with memory, which need to manage inter-symbol interference.

13 PSD Plots

13.1 PSD plot of line Coded signal

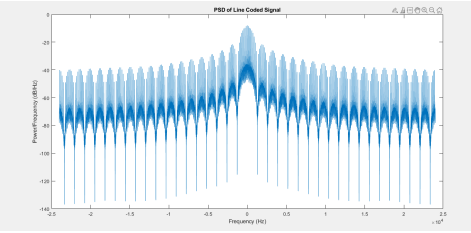

Figure 41: Line Coded PSD

13.2 PSD plot of modulated signal

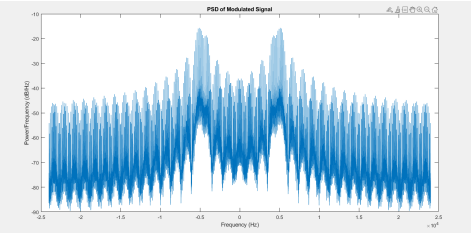

Figure 42: Modulated PSD

13.3 PSD plot of demodulated signal

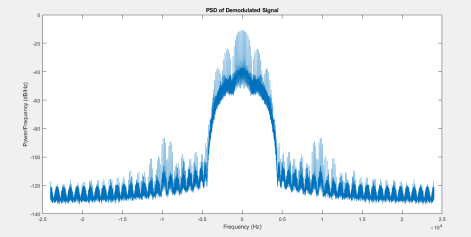

Figure 43: Demodulated PSD

13.4 PSD plot of line decoded signal

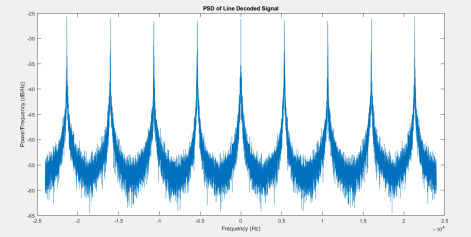

Figure 44: Line Decoded PSD

14 BPSK Calculations and Analysis

14.1 Encoding and Modulation

T Let $b(t)$ denote the data waveform consisting of an infinite sequence of pulses of duration and height ± 1.

$$b(t) = \sum_{l=-\infty}^{\infty} b_l p_T(t - lT), \quad b_l \in \{+1, -1\}.$$

The transmitted signal then is given by

$$\begin{aligned} s(t) &= \sum_{l=-\infty}^{\infty} b_l \cos(2\pi f_c t)\, p_T(t - lT) \\ &= b(t)\cos(2\pi f_c t) = \cos\big(2\pi f_c t + \phi(t)\big) \end{aligned}$$

where $\phi(t)$ is the phase waveform. The signal power is P. The energy of each transmitted bit is $E = PT$.

14.2 Demodulation

The low pass filter (LPF) is a filter "matched" to the baseband signal being transmitted. For BPSK this is just a rectangular pulse of duration T. The impulse response is $h(t) = p_T(t)$. The output of the low pass filter is

$$X(t) = \int_{-\infty}^{\infty} \sqrt{2/T}\cos(2\pi f_c \tau)\, h(t-\tau) r(\tau) d\tau.$$

The sampled version of the output is given by

$$\begin{aligned} X(iT) &= \int_{-\infty}^{\infty} \sqrt{2/T}\cos(2\pi f_c \tau)\, p_T(iT - \tau) r(\tau) d\tau \\ &= \int_{(i-1)T}^{iT} \sqrt{2/T}\cos(2\pi f_c \tau) \Big[\sqrt{2P} b(\tau)\cos(2\pi f_c \tau) \\ &\qquad + n(\tau)\Big] d\tau \\ &= \int_{(i-1)T}^{iT} 2\sqrt{P/T} b_{i-1}\cos(2\pi f_c \tau)\cos(2\pi f_c \tau)\, d\tau + \eta_i. \end{aligned}$$

η_i is Gaussian random variable, mean 0 variance $N_0/2$. Assuming $2\pi f_c T = 2\pi n$ for some integer n (or that $f_c T >> 1$)

$$X(iT) = \sqrt{PT} b_{i-1} + \eta_i = \sqrt{E} b_{i-1} + \eta_i.$$

14.3 Probability of Error

$$P_{e,b} = Q\left(\sqrt{\frac{2E}{N_0}}\right) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

where

$$Q(x) = \int_x^{\infty} \frac{1}{2\pi} e^{-u^2/2} du$$

For binary signals this is the smallest bit error probability, i.e. BPSK are optimal signals and the receiver shown above is optimum (in additive white Gaussian noise). For binary signals the energy transmitted per information bit E_b is equal to the energy per signal E. For $P_{e,p} = 10^{-5}$ we need a bit-energy, E_b to noise density N_0 ratio of $E_b/N_0 = 9.6$ dB. Note: $Q(x)$ is a decreasing function which is $1/2$ at $x = 0$. There

are efficient algorithms (based on Taylor series expansions) to calculate $Q(x)$. Since $Q(x) \leq e^{\{-x^2/2\}}/2$ the error probability can be upper bounded by

$$P_e \leq \frac{1}{2} e^{\{-E_b/N_0\}}$$

which decreases exponentially with signal-to-noise ratio.

14.3.1 Effectiveness

There are two important factors that govern the effectiveness of a digital modulation technique: power efficiency and bandwidth efficiency (also called spectral efficiency). The power efficiency of a digital modulation technique is measured by the E/No ratio required at the receiver for a certain bit error rate. For a given BER power efficiency is how small of a E_p is required. We can compare with using the formula $P_{e,b} = Q\left(\sqrt{\frac{2E}{N_0}}\right)$

14.4 Bandwidth

The power spectral density is a measure of the distribution of power with respect to frequency. The power spectral density for BPSK has the form

$$S(f) = \frac{PT}{2}\left[\text{sinc}^2\left((f - f_c)\,T\right) + \text{sinc}^2\left((f + f_c)\,T\right)\right]$$

where

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

Notice that

$$\int_{-\infty}^{\infty} S(f)df = P$$

The power spectrum has zeros or nulls at $f - f_c = i/T$ except for $i = 0$; that is there is a null at $f - f_c = \pm 1/T$ called the first null; a null at $f - f_c = \pm 2/T$ called the second null; etc. The bandwidth between the first nulls is called the null-to-null bandwidth. For BPSK the null-to-null bandwidth is $2/T$. Notice that the spectrum falls off as $(f - f_c)^2$ as f moves away from f_c. (The spectrum of MSK falls off as the fourth power, versus the second power for BPSK).

It is possible to reduce the bandwidth of a BPSK signal by filtering. If the filtering is done properly the (absolute) bandwidth of the signal can be reduced to $1/T$ without causing any intersymbol interference; that is all the power is concentrated in the frequency range

15 BER vs SNR plots

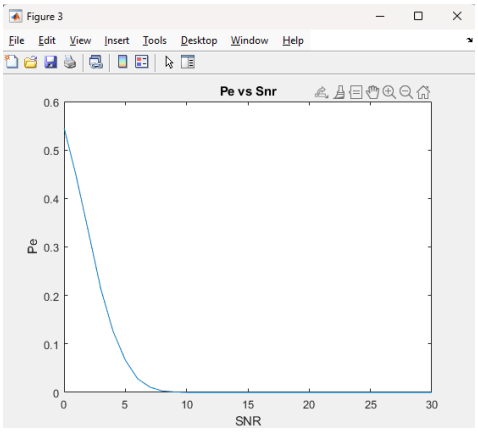

Figure 45: plot for noise with memory

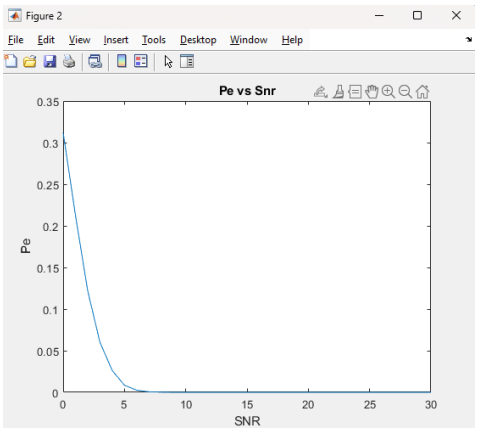

Figure 46: plot for noise

16 Constellations

16.1 Input Constellation

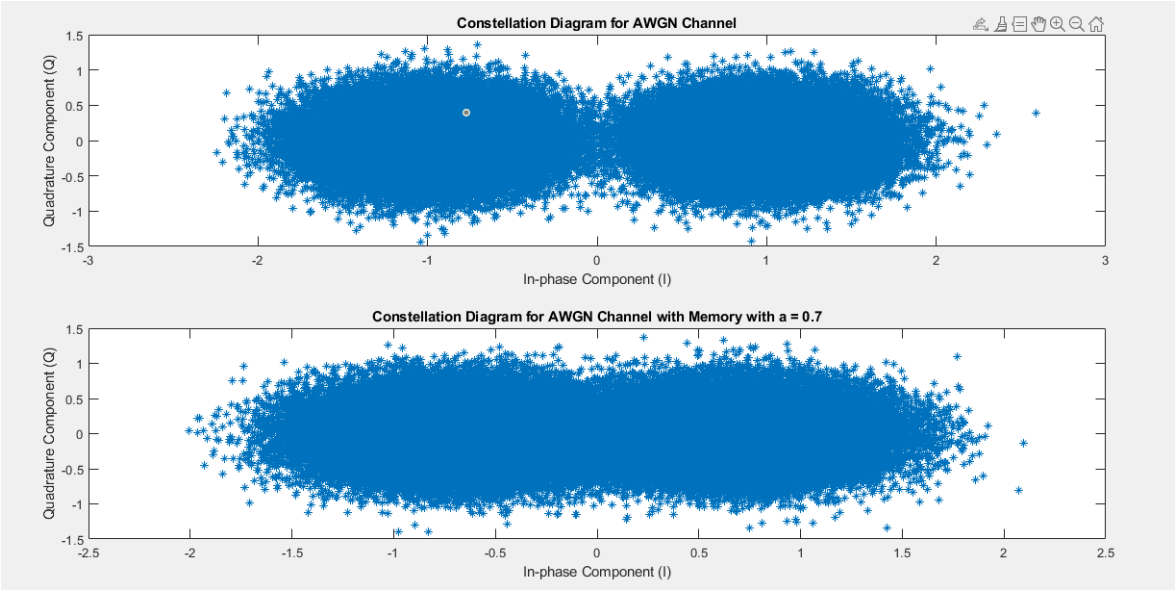

Figure 47: constellation with a=0.7

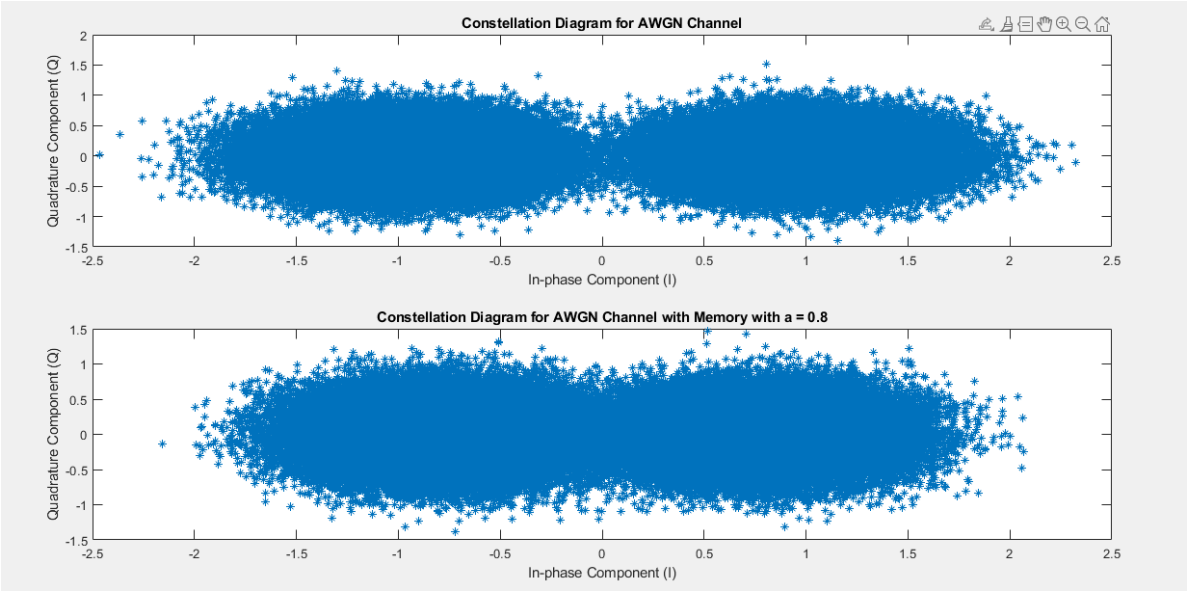

Figure 48: constellation with a=0.8

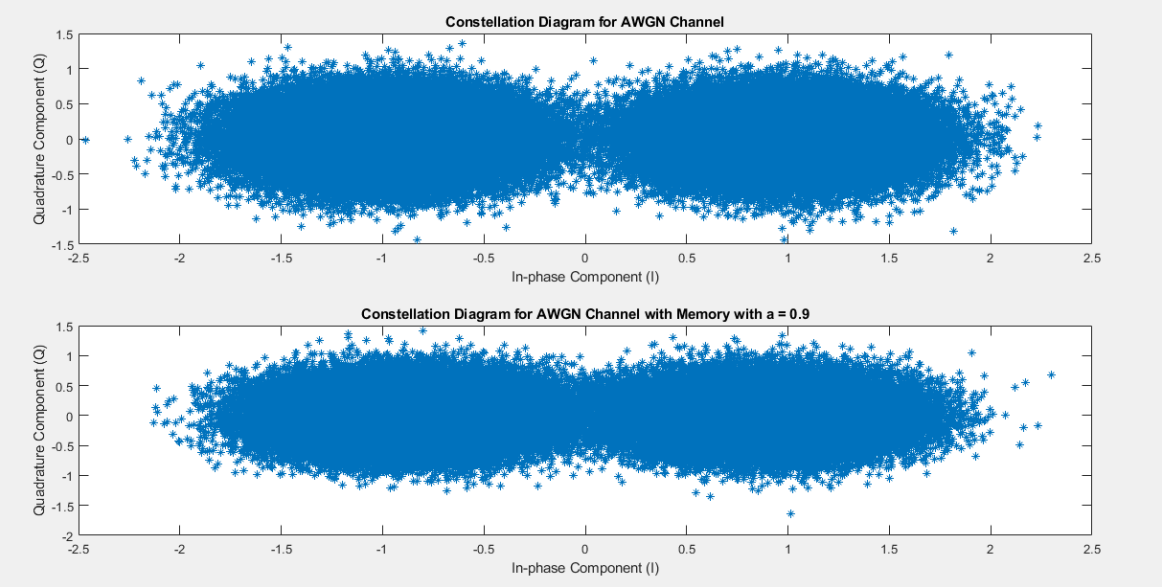

Figure 49: constellation with a=0.9

16.2 Output Constellation with varying a

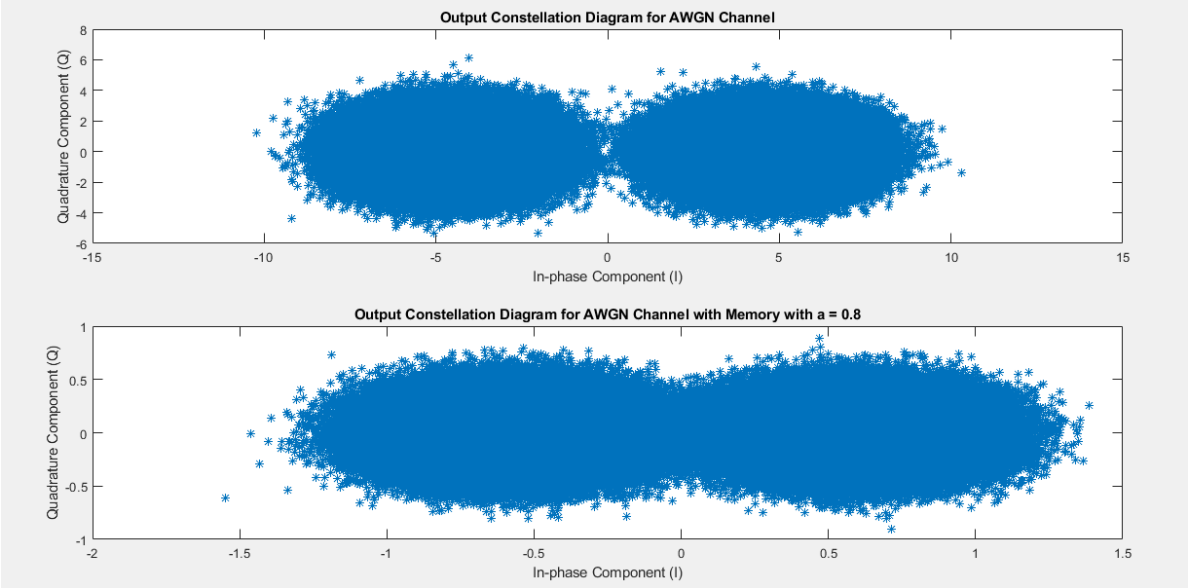

Figure 50: constellation with a=0.8

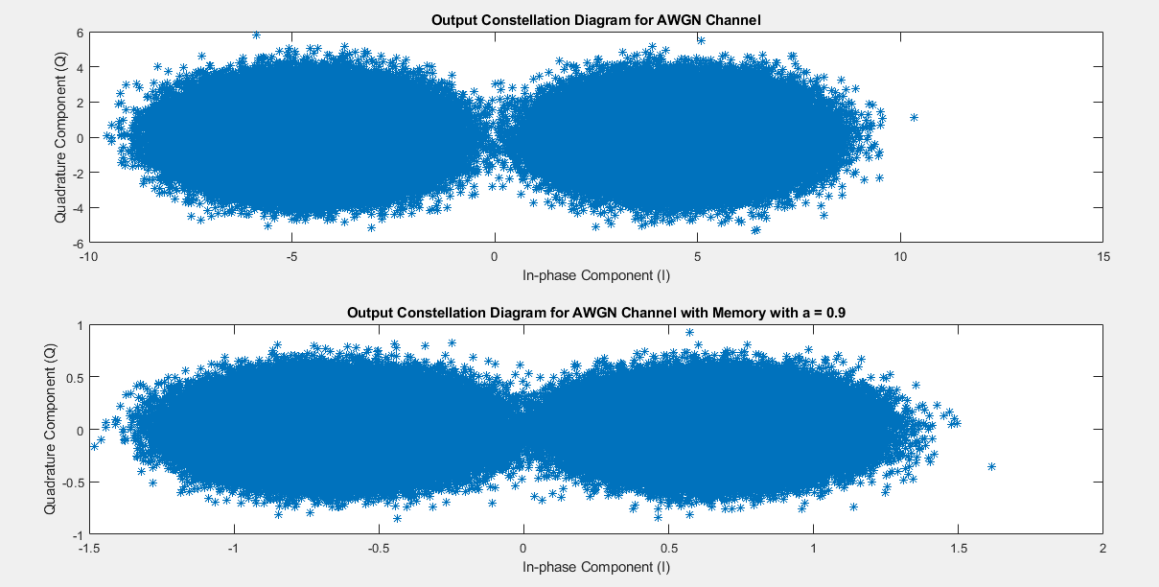

Figure 51: constellation with a=0.9

16.3 Input Constellation with varying b

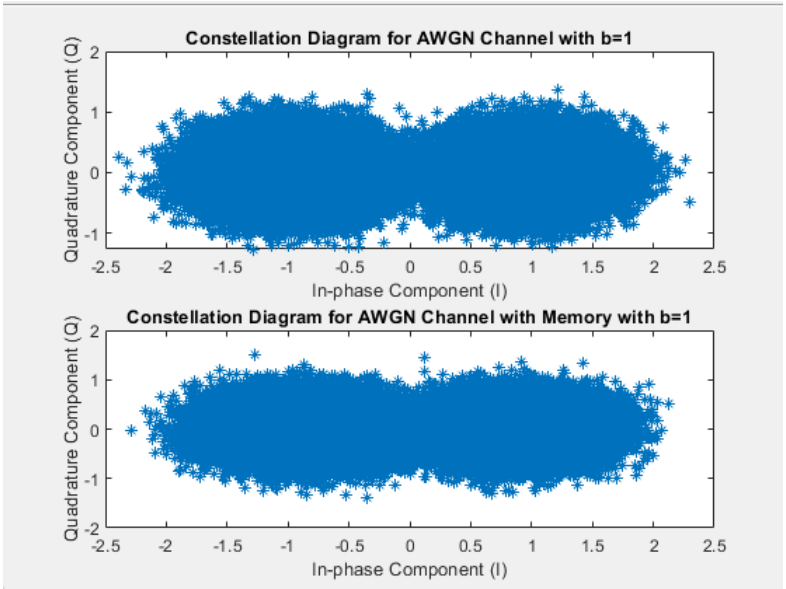

Figure 52: constellation with b=1

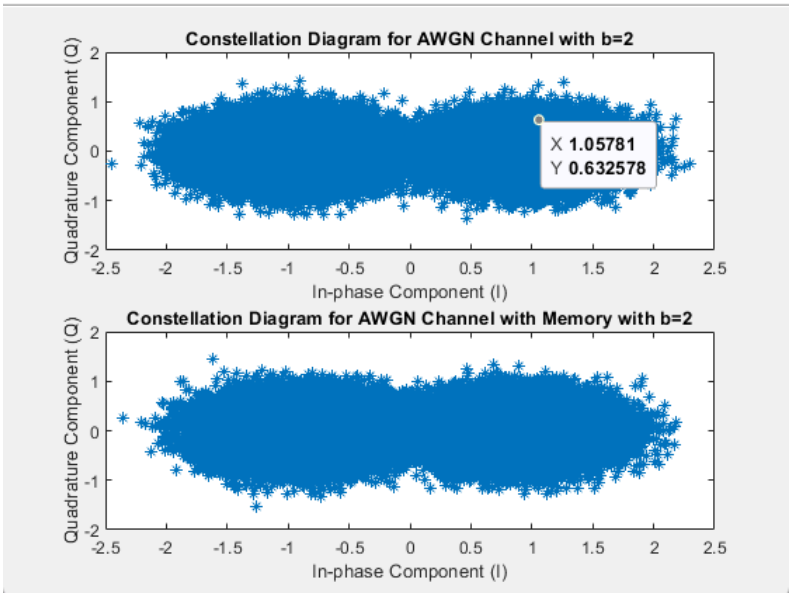

Figure 53: constellation with b=2

16.4 output Constellation with varying b

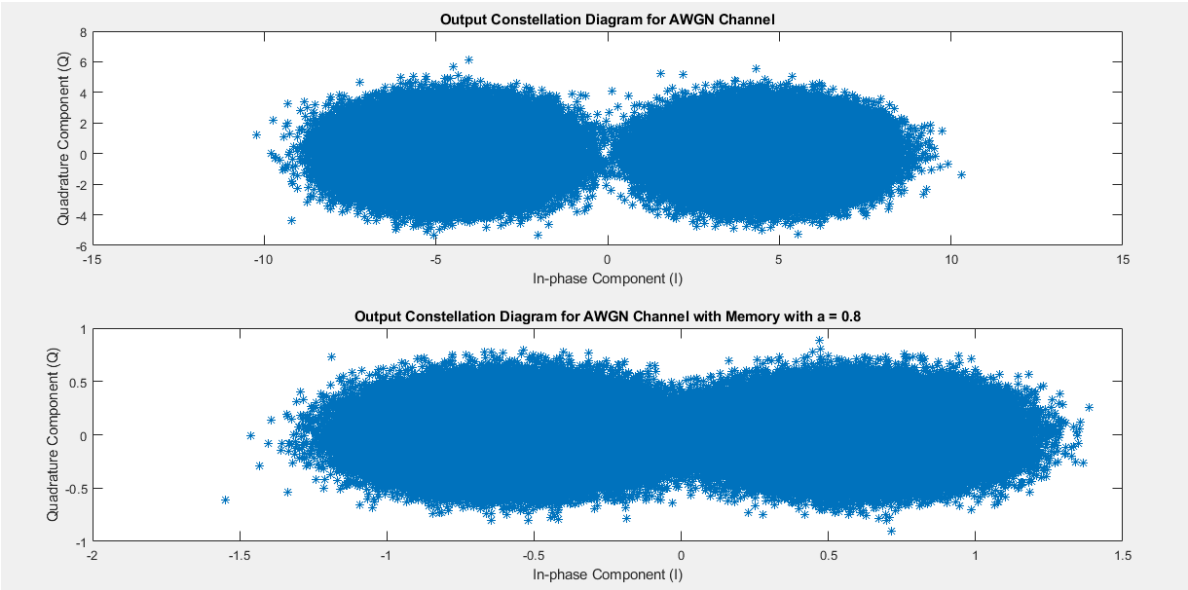

Figure 54: constellation with b=1

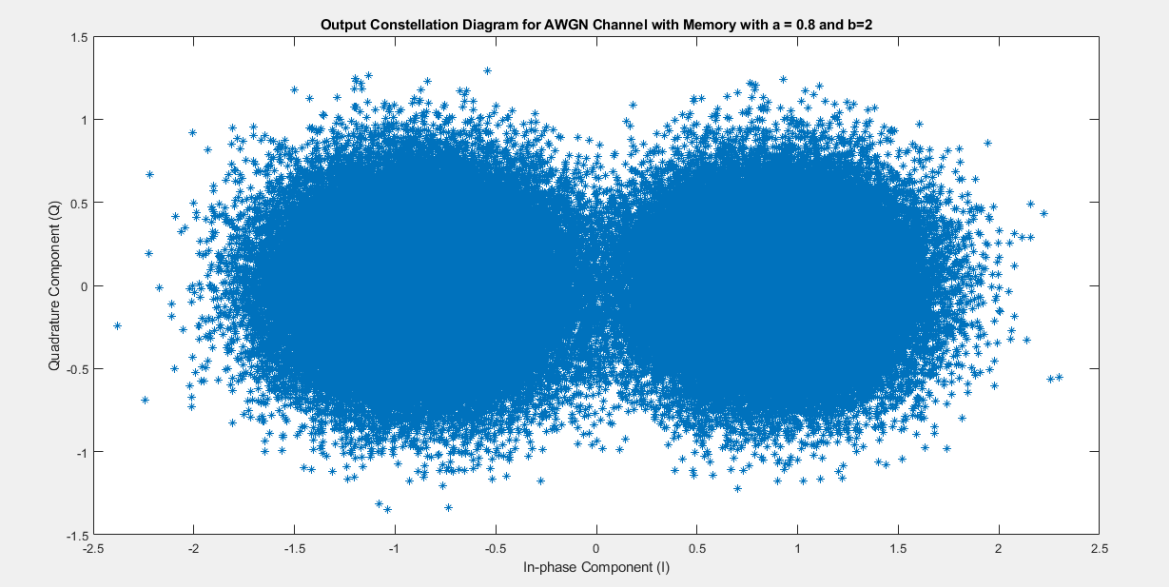

Figure 55: constellation with b=2