Assignment 2, Digital communication 1TE747,VT24

Anton Blaho Mildton
Department of Electrial engineering
Uppsala University
Anton.blahomildton@gmail.com

Gustaf Löfdahl
Department of Electrial engineering
Uppsala University
Gustaf.lofdahl@hotmail.com

UPPSALA
UNIVERSITET

1 Introduction

In the era of fast technological advancement, digital communication has emerged as a transformative force. Unlike traditional analog methods, digital communication relies on discrete signals, enabling more efficient and reliable data transmission. This shift has changed how we connect, share information, and collaborate globally. In the second segment of the Digital Communication course, the focus lies on essential concepts such as converting bitstreams to M-ary PAM-signal. Also implement a demodulation part which demodulate the PAM signal into estimatedbitstream, with and without a matched filter. This is within assignment 2 of the course.

2 Assignment 2 answers

2.1 Subtask 1

First task off the assignment was to create the function in equation 1. The function's goal is to convert a bit stream to a M-ary baseband signal.

$$transmitSignal = MyMPAM(bitstream, M, Es) \tag{1}$$

The input of the function is the original bitstream, which can be created in assignment 1. M represents the number of different amplitude levels for both in-phase and quadrature components, and Es is the average energy of the symbol levels.

The transmitted signal with M=8, which equals 1 bit, according to the equation 2, is shown in figure 1

$$M = 2^k \tag{2}$$

Where k is the amount of bits. Equation 3 is the formula to calculate Es, where d is distance between levels.

$$\bar{E_s} = \frac{1}{M}\sum_{i=1}^{M} A_i^2 = \frac{1}{M}\sum_{i=1}^{M}(2i-1-M)^2 d^2 = \frac{1}{3}(M^2-1)d^2 \tag{3}$$

Solving for d yields:

$$\frac{3\overline{E}}{M^2-1} = d \tag{4}$$

Each PAM-symbol s_i in $transmitSignal$ is created by:

$$s_i(t) = A_i g(t)\cos(2\pi f_c t) \tag{5}$$

For baseband-PAM, $f_c = 0$, and equation (5) becomes $s_i(t) = A_i g(t)$.

2.2 Subtask 2

In the second subtask the goal was to create function in equation 6. This function has the received PAM signal (Equation 1) as input togheter with M, Es and the transmittedbitstream.

$$[\text{estimatedBitstream}, \text{BER}] = \text{DemodulateMPAM}(\text{receivedSignal}, M, Es, \text{transmittedBitstream}) \tag{6}$$

This function should demodulate the MPAM signal every T sample, and compare them with the symbols to perform a estimation of the original bitstream. Additionally, it should calculate the empirical BER, which stands for Bit Error Rate (BER).BER gives an idea of how well the received signal matches the transmitted signal in terms of bit accuracy. To see that the original bitstream matched with estimatedbitstream, a random segment of the estimatedbitstream has randomly picked out to see of it matched with the original bitstream.

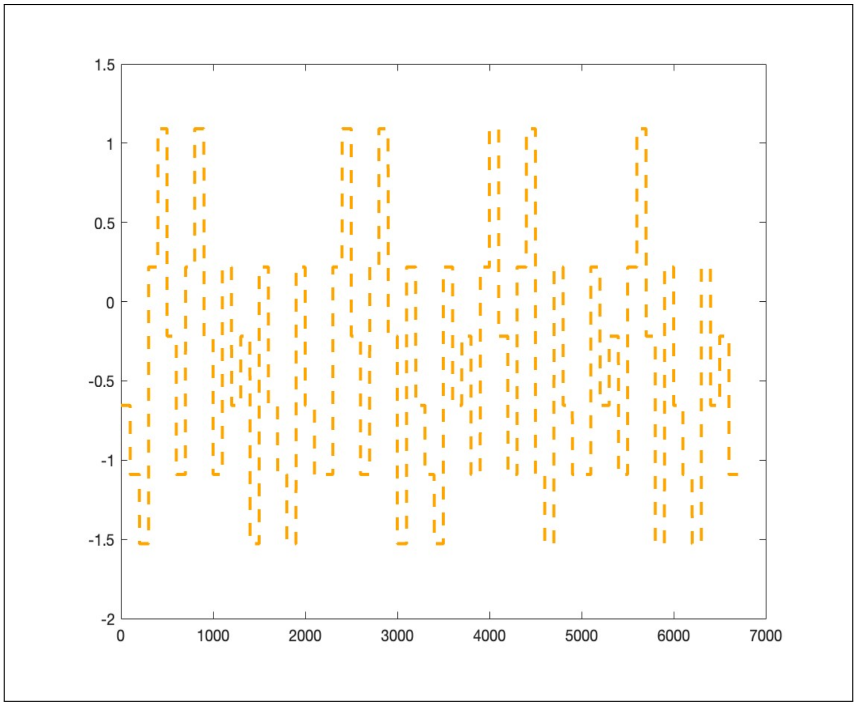

Figure 1: The transmitted signal (train) with M=8, Es=1 and sampling time 100.

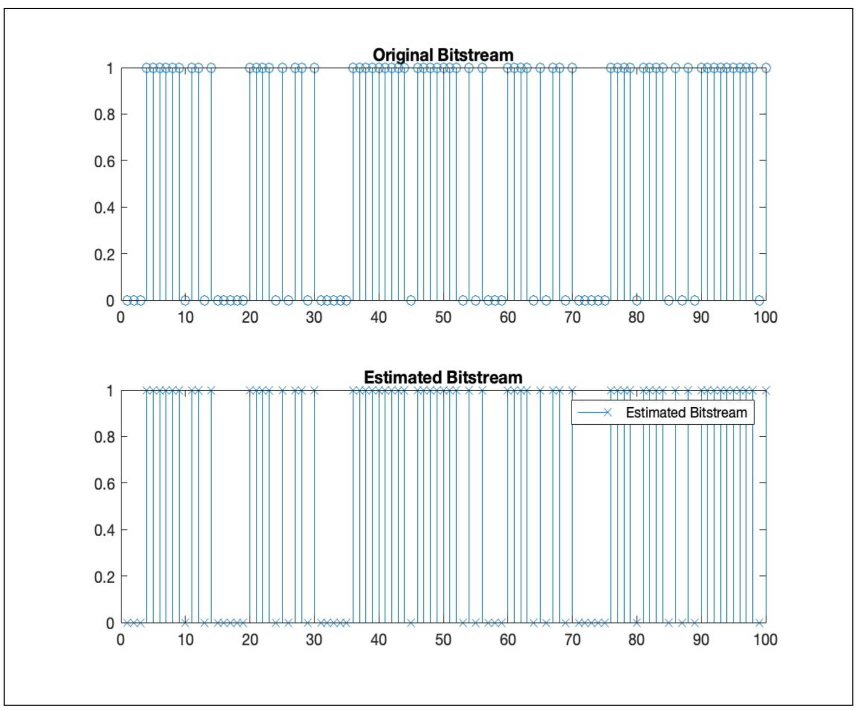

Figure 2: Original vs estimated bitstream segment.M=2

2.3 Subtask 3

Here, the task was to provide an ilustration of the BER that it is zero when there is no noise applied to the signal. This means that all the bits are correctly received. The code in listing 1 provides the code for the subtask where the user call the function and figure 3 provides the resulting displayed BER for $M = 2, 4, 8$.

```
%Calling the functions
spacing=100;

Ber_2_PAM=zeros(spacing,1);
Ber_8_PAM=zeros(spacing,1);

%transmitSignal = MyMPAM(bitstream, M, Es);
%receiveSignal = MyAWGNchannel(transmitSignal,0.01);
[estimatedBitStream, BER] = DemodulateMPAM(MyMPAM(bitstream,M,Es), M,
    Es, bitstream, matchedFilterFlag);



fprintf('Bit Error Rate (BER): %.4f\n', BER);
```

Listing 1: MATLAB code

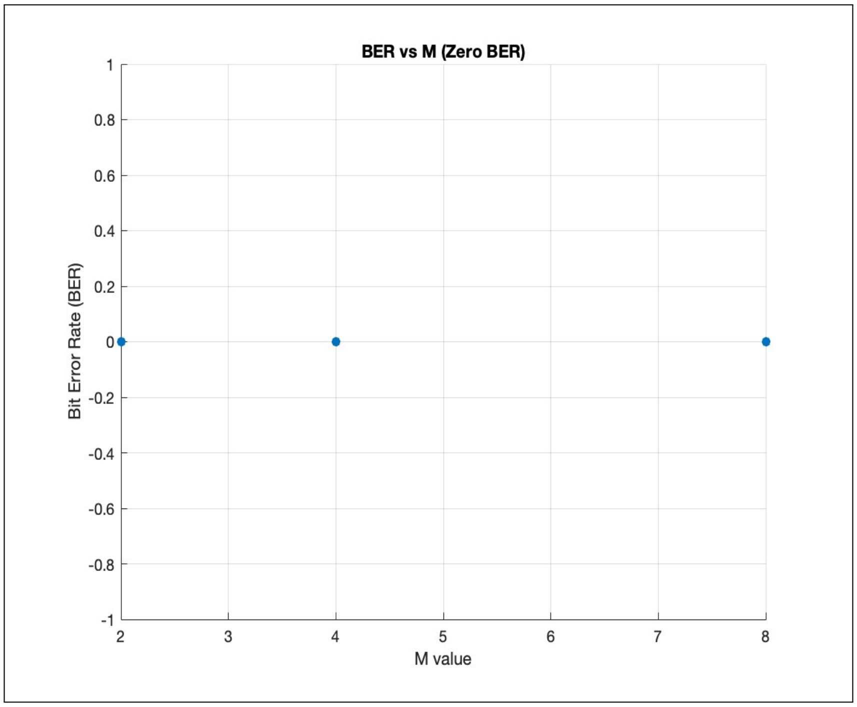

Figure 3: BER value with no noise applied, M=8

As can be seen the Bit error Rate is 0 with no noise applied.

2.4 Subtask 4

In this subtask a function should be created that adds noise to to the channel. Equation 7 is the function that should be created.

$$\text{receiveSignal} = \text{MyAWGNchannel}(\text{transmitSignal}, \text{noiseVariance}) \tag{7}$$

By entering the nosie variance in a vector, created in MATLAB with the logspace function. The following plots illustrate the BER dependant on the noise variance for 2PAM and 8PAM. This plots follows down below in figure 4 and 5. Additionally, a plot on the noise signal against the true transmitted signal is shown in figure 6. In figure 6 blue is the noise signal and orange is the transmitted signal. It is important to note that it is just a small part of the signal.

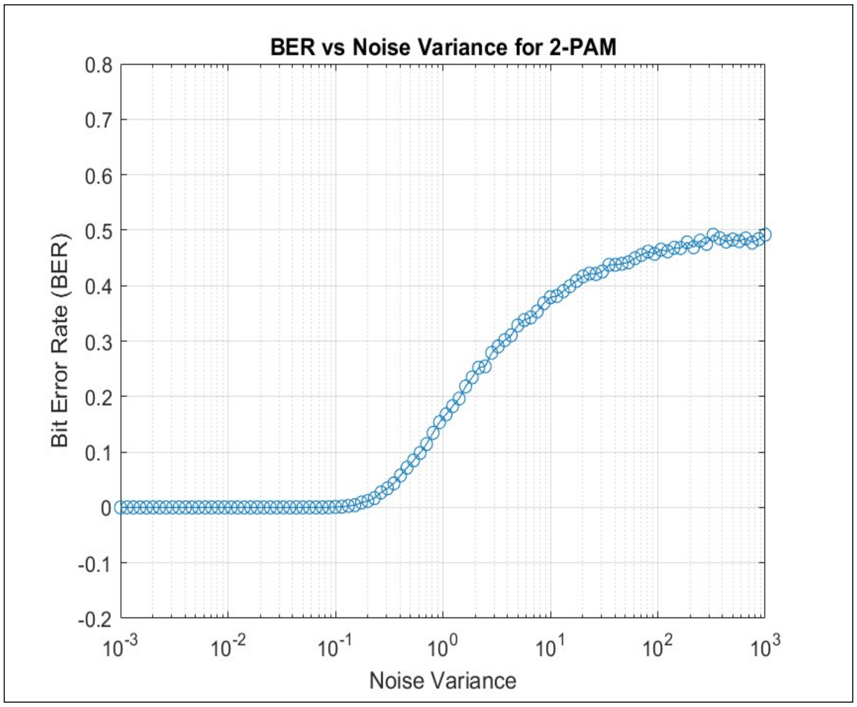

Figure 4: BER against noise variance, 2PAM

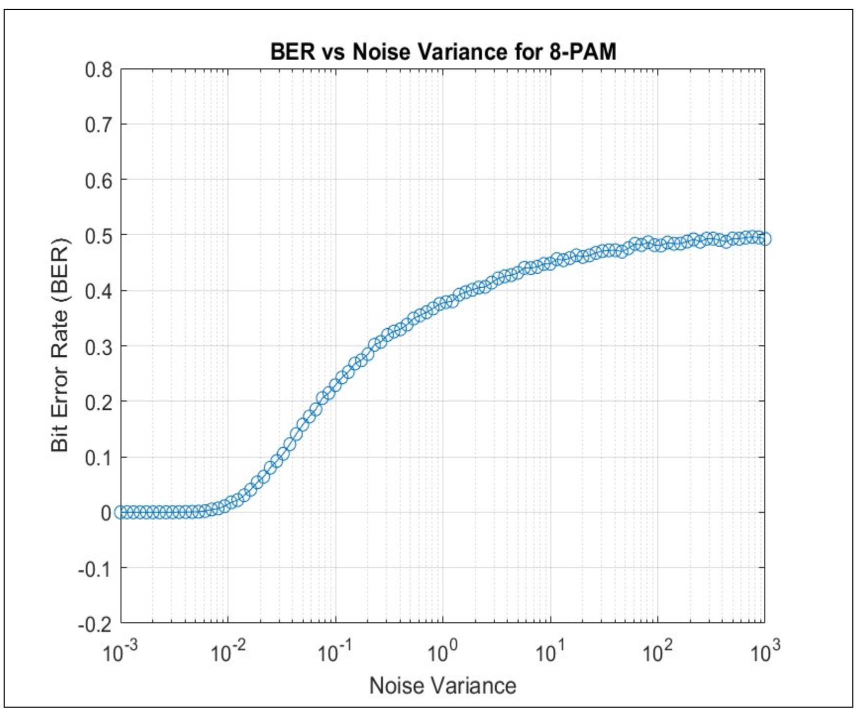

Figure 5: BER against noise variance, 8PAM

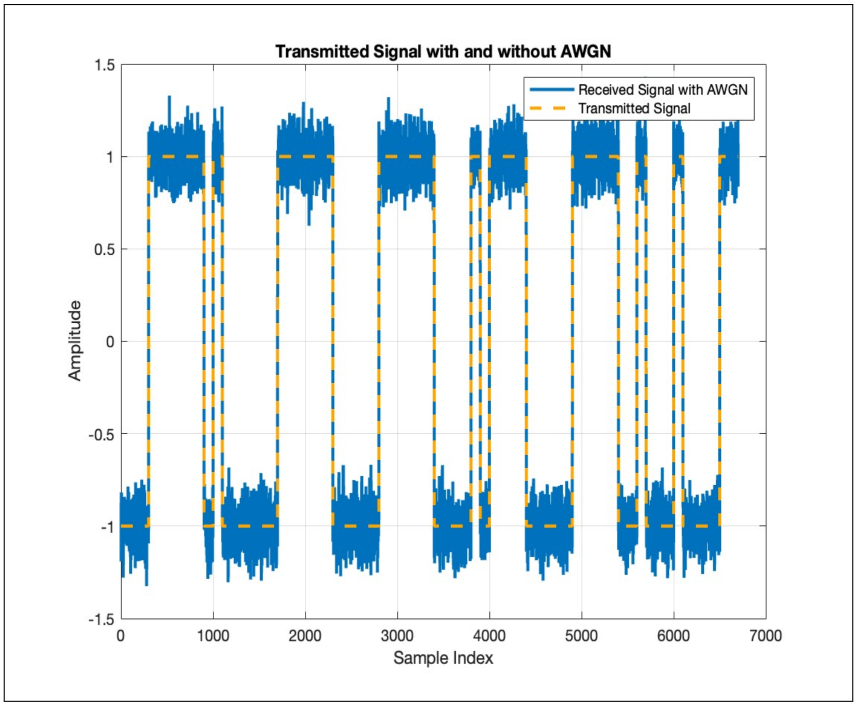

Figure 6: Transmitted signal without noise against signal with noise

2.5 Subtask 5

The task here is to derive the expression of the output of the matched filter. Extensive discussion of the matched filter can be found on page 410 of A. Goldsmith's book on Wireless Communication. The expression for the output of the matched filter in the receiver is as follows:

$$\hat{s}_i = \int_0^{T_s} \hat{x}_t^* * g^*(-t)\, dt$$

Where $g^*(-t)$ signifies the matched filter. In correspondence related to the receiver structure, the equation is analogous to:

$$\hat{s}_i = K \int_0^{T_s} \hat{x}_t g(t)\, dt$$

Since $\hat{x}_t = s_i = A_i \cdot g(t) \cdot \cos(2\pi f_c t)$ (the noise-free signal coming from the transmitter) and $f_c = 0$, the above equation gives:

$$\hat{s}_i = K \int_0^{T_s} A_i g(t)\, dt$$

Since $g(t) = 1$ during the symbol time T_s, the equation for $\hat{s}_i$ becomes:

$$\hat{s}_i = K A_i T_s$$

For $\hat{s}_i$ (the estimate of the symbol s_i being input to the transmitter) to be equal to s_i, KT_s has to equal 1. This yields:

$$K = \frac{1}{T_s}$$

2.6 Subtask 6

In this task the function in the following equation should be constructed. The goal is to implement a matched filter in the demodulation part.

$$
\begin{aligned}
&[\texttt{estimatedBitstream}, \texttt{BER}] = \texttt{DemodulateMPAM} \\
&\qquad\qquad (\texttt{receivedSignal}, M, Es, \texttt{transmittedBitstream}, \texttt{matchedFilterFlag})
\end{aligned} \tag{8}
$$

Here what is added is the matchedFilterFlag, which is supposed to be 1 if it is used and 0 if it is not used. The code (on how the matched filter was implemented) for this part can be seen in listing 2 but also in the appendix A. The same plots as in subtask 4 (BER against noise variance) should be illustrerad here, with the difference that this time is the matched filter used for the demodulation. This figures can be seen in figure 7 and 8.

```
function [estimatedBitStream, BER] = DemodulateMPAM(receivedSignal, M,
     Es, transmittedBitstream, matchedFilterFlag)

T=100;
d = sqrt(3*Es/(M^2-1));
k = log2(M);
receivedMatrix=receivedSignal;

if matchedFilterFlag==1
    segmentSize=floor(length(receivedMatrix)/T);
    integral1=zeros(1,segmentSize);
    for j=1:segmentSize
        integral1(j)=trapz(receivedSignal((1+((j -1)*T)):(j*T)))/T ;
    end
else
    receivedMatrix = receivedMatrix;
    receivedMatrix = downsample(receivedSignal, T).';
    integral1=receivedMatrix.';

end
```

Listing 2: MATLAB code for matched filter

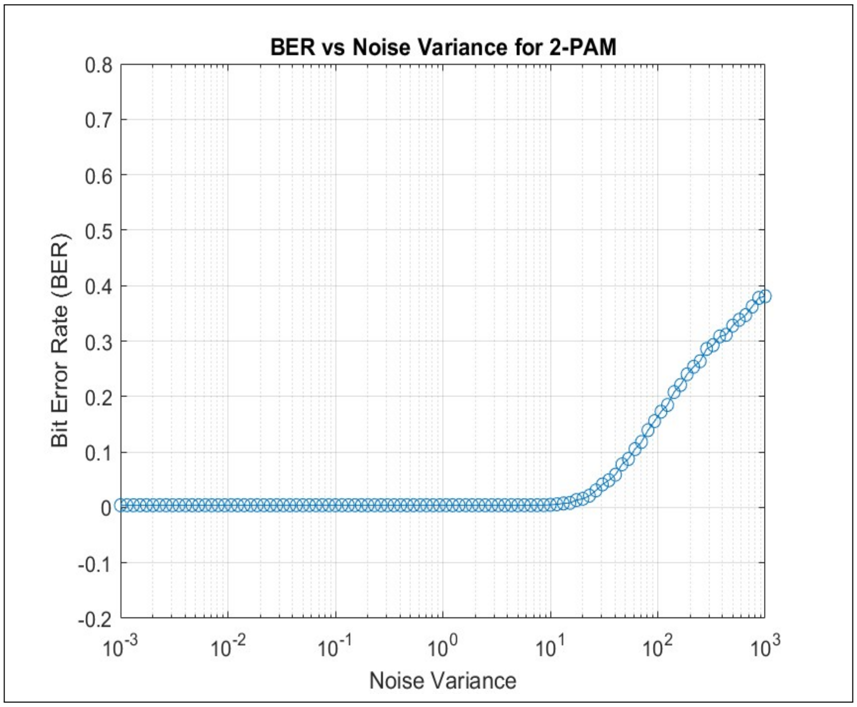

Figure 7: BER against noise variance with matched filter,2PAM

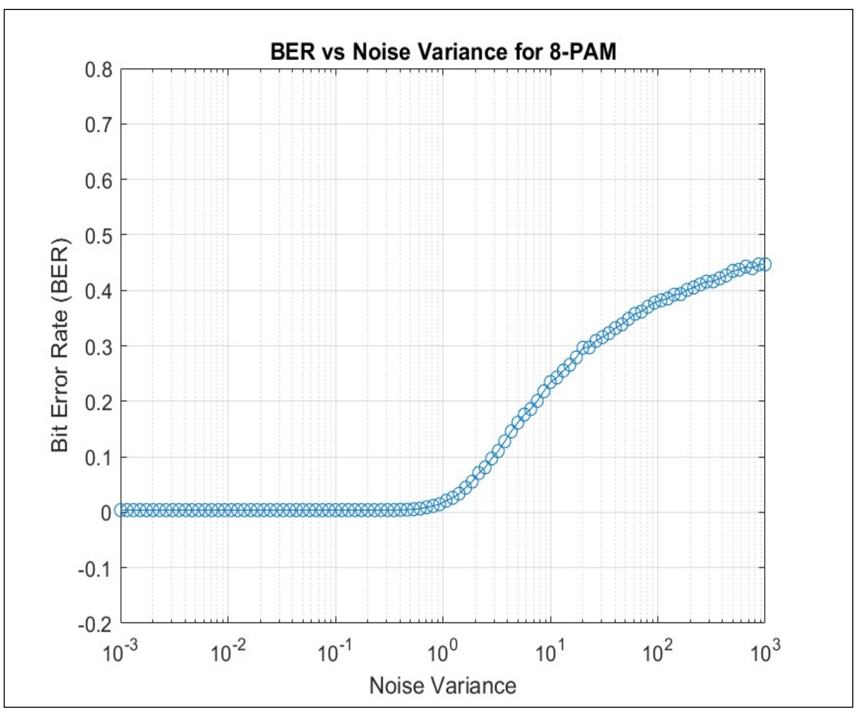

Figure 8: BER against noise variance with matched filter, 8PAM

As can be seen in these figures, the BER is shifted by a factor of 10^2 on the noise variance axis if the matched filter is applied compared to without. Meaning that the matched filter is able to correctly determine the bits with noise with higher variance that is demodulated incorrectly when the matched filter is not applied. An observation was also done when the noise is increased significantly 10^3, then the BER will increase to the same values without the matched filter. This means that the filter is better working when the noise is around 10^{-3} and 10^0.

2.7 Subtask 7

This task was to implement everything in assignment 1 to assignment 2. The code for this can be seen in appendix A. In this report all figures are based on the train signal used in assignment 1 and the estimatedbistream from this assignment. All the figures shown in this report is a verification of that it works.

References

[1] *Wireless Communication, Andrea Goldsmith,* Available at: http://fa.ee.sut.ac.ir/Downloads/AcademicStaff/1/Courses/7/Andrea%20Goldsmith-Wireless%20Communications-Cambridge%20University%20Press%20%282005%29.pdf Accessed: January 29, 2024.

A Appendix

```
clc
clearvars
close all

load train;
unquantizedSignal = y;
matchedFilterFlag = 1;
Vp = 1;
N = 3;
Es = 1;
M = 2^N;

% From Assignment 1
[quantizedSignal,~,~,~,~,~] = MyQuantizer(unquantizedSignal,Vp,N);
transmitedBitStream = MyGraycode(quantizedSignal,Vp,N);

% Assignment 2
transmitSignal = MyMPAM(transmitedBitStream,M,Es);
receiveSignal = MyAWGNchannel(transmitSignal,0.01);
[estimatedBitStream,BER1] = DemodulateMPAM(receiveSignal, M, Es,
    transmitedBitStream, matchedFilterFlag);
estimatedSignal = MyDAconverter(estimatedBitStream,Vp,N);

% Random index for plots
sx = randi(length(y) -1000);

figure
% Plot train signal
plot(unquantizedSignal)
title("Unquantized Signal")
ylabel("Amplitude (V)")
xlabel("Time step (1/Fs)")

figure
% Plot quantized train signal
plot(quantizedSignal)
title(['Quantized Signal, Vp = ',num2str(Vp),', N = ',num2str(N)])
ylabel("Amplitude (V)")
xlabel("Time step (1/Fs)")

figure;
% Plot received signal with AWGN
plot(receiveSignal(sx:sx+10000), 'LineWidth', 2, 'DisplayName', '
    Received Signal with AWGN');
hold on;
% Plot transmitted signal without AWGN with transparency
plot(transmitSignal(sx:sx+10000), 'LineWidth', 2, 'DisplayName', '
    Transmitted Signal', 'Color',[1 0.6471 0], 'LineStyle', '--');
hold off;
xlim([0,10000])
xlabel('Sample Index');
ylabel('Amplitude');
title('Transmitted Signal with and without AWGN');
legend('show');
grid on;
```

```matlab

figure;
% Plot transmitted bitstream without AWGN
subplot(2, 1, 2);
stem(estimatedBitStream(sx:sx+100), 'Marker', 'x', 'DisplayName', 'Estimated Bitstream');
title('Estimated Bitstream');
xlim([0,100])
% Plot estimated bitstream with AWGN
subplot(2, 1, 1);
stem(transmitedBitStream(sx:sx+100), 'Marker', 'o', 'DisplayName', 'Original Bitstream');
title('Original Bitstream');
xlim([0,100])

% Clearing for BER calculations
clearvars
load train;
unquantizedSignal = y;
matchedFilterFlag = 1;
spacing = 100;
noiseVariance = logspace(-3,3,spacing);
Vp = 1;

% Calculating BER of 2PAM
N = 1;
Es = 1;
M = 2^N;
[quantizedSignal,~,~,~,~,~] = MyQuantizer(unquantizedSignal,Vp,N);
transmitedBitStream = MyGraycode(quantizedSignal,Vp,N);

for i = 1:spacing
    receivedSignal = MyMPAM(transmitedBitStream, M, Es);
    receiveSignal = MyAWGNchannel(receivedSignal,noiseVariance(i));
    [~, BER] = DemodulateMPAM(receiveSignal, M, Es, transmitedBitStream, matchedFilterFlag);
    Ber_2_PAM(i) = BER;
end

% Calculating BER of 2PAM
N = 3;
Es = 1;
M = 2^N;
[quantizedSignal,varLin,varSat,varTeo,SNqR,SNqRTeo] = MyQuantizer(unquantizedSignal,Vp,N);
transmitedBitStream = MyGraycode(quantizedSignal,Vp,N);

for i = 1:spacing
    receivedSignal = MyMPAM(transmitedBitStream, M, Es);
    receiveSignal = MyAWGNchannel(receivedSignal,noiseVariance(i));
    [~, BER] = DemodulateMPAM(receiveSignal, M, Es, transmitedBitStream, matchedFilterFlag);
    Ber_8_PAM(i) = BER;
end

% Plotting log noise
figure;
loglog(1:spacing, noiseVariance, 'o-', 'LineWidth', 2);
title('Noise Variance vs. Index of Columns');
xlabel('Index of Columns');
ylabel('Noise Variance');
grid on;

% Plot BER of 2PAM
figure;
```

```
semilogx(noiseVariance, Ber_2_PAM, '-o');
title('BER vs Noise Variance for 2-PAM');
xlabel('Noise Variance');
ylabel('Bit Error Rate (BER)');
xlim([0.001,1000]);
ylim([-0.2,0.8]);
grid on;

% Plot BER of 8PAM
figure;
semilogx(noiseVariance, Ber_8_PAM, '-o');
title('BER vs Noise Variance for 8-PAM');
xlabel('Noise Variance');
ylabel('Bit Error Rate (BER)');
xlim([0.001,1000]);
ylim([-0.2,0.8]);
grid on;

% Functions

% Asignment 1 Functions
function [quantizedSignal,varLin,varSat,varTeo,SNqR,SNqRTeo] = MyQuantizer(unquantizedSignal,Vp,N)
    %quantization setup
    levels = 2^N;
    step = 2*Vp/levels;
    varTeo = step^2/12;
    SNqRTeo = mag2db(levels^2);
    level = linspace(-(levels/2-0.5)*step,(levels/2-0.5)*step,levels);
    quantizedSignal = nan(1,length(unquantizedSignal));

    %quantization
    for i = 1:length(unquantizedSignal)
        for j = 1:length(level)
            if unquantizedSignal(i) <= level(j)+step/2 && unquantizedSignal(i) > level(j)-step/2
                quantizedSignal(i) = level(j);
            end
        end
        if unquantizedSignal(i) < level(1)
                quantizedSignal(i) = level(1);
        elseif unquantizedSignal(i) > level(length(level))
                quantizedSignal(i) = level(length(level));
        end
    end

    %Variance linear
    varLin=var(quantizedSignal.' - unquantizedSignal);

    % Saturated error variance
    satError = quantizedSignal.' - unquantizedSignal;
    satError = min(max(satError, -Vp), Vp);
    varSat = var(satError);

    % Signal to Quantization Noise power Ratio (SNqR) in dB
    SNqR = 20 * log10(var(unquantizedSignal) ./ varSat);
end

function [bitStream] = MyGraycode(quantizedSignal,Vp,N)
    levels = 2^N;
    step = 2*Vp/levels;
    bitStream = nan(1,N*length(quantizedSignal));
    for i = 1:length(quantizedSignal)
        bit = (quantizedSignal(i)+(levels/2-0.5)*step)/step;
        bin = dec2bin(bit,N);
```

```
        bitStream(1,1+N*(i-1)) = str2double(bin(1));
        for j = 2:N
            bitStream(1,j+N*(i-1)) = bitxor(str2double(bin(1,j)), str2double(bin(1,j-1)));
        end
    end
end

% Asignment 2 Functions
function transmitSignal = MyMPAM(bitstream,M,Es)
    d = sqrt(3*Es/(M^2-1));
    k = log2(M);
    symbolMatrix = reshape(bitstream, k, length(bitstream)/k)';

    % Map symbols to amplitudes (equally and symmetrically spaced)
    amplitudeLevels = linspace(-(M-1)*d, (M-1)*d, M);
    symbolVector = zeros(1,length(symbolMatrix));
    for i = 1:length(symbolVector)
        for j = 1:k
            symbolVector(i) = symbolVector(i) + symbolMatrix(i,j)*2^-(j-k);
        end
    end
    transmitSignalfirst = amplitudeLevels(symbolVector+1);

    % Rectangular pulse shaping
    sampleRate = 100;
    shapedSignal = rectpulse(transmitSignalfirst, sampleRate);

    % Return shaped signal
    transmitSignal=shapedSignal;
end

function receiveSignal = MyAWGNchannel(transmitSignal, noiseVariance)
    % Generate white Gaussian noise with specified variance
    noise = wgn(size(transmitSignal, 1), size(transmitSignal, 2), noiseVariance, 'linear');

    % Add noise to the transmitted signal
    receiveSignal = transmitSignal + noise;
end

function [estimatedBitStream, BER] = DemodulateMPAM(receivedSignal, M, Es, transmittedBitstream, matchedFilterFlag)
T = 100;
d = sqrt(3*Es/(M^2-1));
k = log2(M);
receivedVector = receivedSignal;
if matchedFilterFlag == 1
    %amplitudeLevels = linspace(-(M-1)*d, (M-1)*d, M);
    segmentSize=floor(length(receivedVector)/T);
    integral=zeros(1,segmentSize);
    for j=1:segmentSize-99
        integral(j)=trapz(receivedSignal((1+((j -1)*T)):(j*T)))/T ;
    end
else
    receivedVector = downsample(receivedSignal, T).';
    integral=receivedVector.';
end

% Estimate symbols from received samples
amplitudeLevels = linspace(-(M-1)*d, (M-1)*d, M);
estimatedSymbols = zeros(1,size(integral, 2));
for i = 1:size(integral, 2)
    [~, index] = min(abs(integral(1, i) - amplitudeLevels));
```

```
    estimatedSymbols(i) = index-1;
end

% Convert symbols to bitstream
estimatedSymbolMatrix = de2bi(estimatedSymbols, k, 'left-msb');
estimatedBitStream = reshape(estimatedSymbolMatrix',1, []);

% Calculate Bit Error Rate (BER)
numErrors = sum(estimatedBitStream ~= transmittedBitstream);
BER = numErrors / length(transmittedBitstream);
end

% Asignment 1 Function
function [estimatedSignal] = MyDAconverter(estimatedBitStream,Vp,N)
    levels = 2^N;
    step = 2*Vp/levels;
    level = linspace(-(levels/2-0.5)*step,(levels/2-0.5)*step,levels);
    signalbits = nan(1,length(estimatedBitStream));
    estimatedSignal = zeros(1,length(estimatedBitStream)/N);
    for i = 1:length(estimatedSignal)
        signalbits(1+N*(i-1)) = estimatedBitStream(1+N*(i-1));
        for j = 2:N
            signalbits(j+N*(i-1)) = bitxor(estimatedBitStream(j+N*(i
    -1)),signalbits(j+N*(i-1)-1));
        end
        for k = 1:N
             estimatedSignal(i) = estimatedSignal(i) + signalbits(k+N
    *(i-1))*2^(N-k);
        end
    end
    for l = 1:length(estimatedSignal)
        estimatedSignal(l) = level(estimatedSignal(l)+1);
    end
end
```

Listing 3: MATLAB code

```
%Calling the functions
spacing=100;

Ber_2_PAM=zeros(spacing,1);
Ber_8_PAM=zeros(spacing,1);

%transmitSignal = MyMPAM(bitstream, M, Es);
%receiveSignal = MyAWGNchannel(transmitSignal,0.01);
[estimatedBitStream, BER] = DemodulateMPAM(MyMPAM(bitstream,M,Es), M,
    Es, bitstream, matchedFilterFlag);



fprintf('Bit Error Rate (BER): %.4f\n', BER);
```

Listing 4: MATLAB code